

Universidad Carlos III de Madrid
Escuela Politécnica Superior
Ingeniería de Telecomunicación



Proyecto Fin de Carrera

**Máquinas de vectores soporte
para procesamiento de secuencias:
el kernel de Fisher
en secuencias bioinformáticas**

Autor: Pablo Barrera González
Tutor: Dr. Ángel Navia Vázquez
Noviembre 2002

*A mis padres,
Benito y Esther,
y a todas mis mujeres,
Amaya, Tamara y Ruth*

Agradecimientos

Al finalizar este trabajo, me gustaría agradecer a todos los que, día a día, me han ayudado a conseguir terminar este viaje, que empezó hace ya bastante tiempo.

En primer lugar, quiero comenzar por mi familia, porque ellos me han apoyado desde el primer momento, con su cariño y comprensión. Gracias a mis hermanas, mis abuelos, mis tíos y tías, pero sobre todo a mis padres, porque sin ellos nunca podría haber llegado hasta dónde he llegado.

También a los compañeros, con los que he compartido penas y pesares, clase tras clase, crédito tras crédito; porque más que compañeros han sido, son y serán, amigos. Gracias a ellos hubo muchos más buenos momentos que malos. En especial a Javi, Guille, Sergio y Carlos, porque siempre han estado ahí.

En cuanto a la realización de este proyecto en particular, me gustaría agradecer su ayuda a la Doctora Ascensión, por su preocupación y sus consejos, que siempre fueron muy útiles para mí.

Por último, quiero agradecerte todo esto, y mucho más, a ti, Ruth; por ser quien eres, por ser como eres, por estar ahí y por estar aquí. Porque los proyectos que realmente son importantes son los que empiezan contigo, porque cualquier cosa a tu lado es mucho más fácil.

ÍNDICE GENERAL

I	Proyecto Fin de Carrera	1
1.	Introducción	3
1.1.	Estado del arte	3
1.2.	Objetivos	5
1.3.	Organización del proyecto	5
2.	Kernel de Fisher	7
2.1.	Aprendizaje máquina	7
2.1.1.	Clasificación y regresión	8
2.2.	Máquinas de vectores soporte	9
2.3.	Motivación del Kernel de Fisher	11
2.4.	La función de kernel de Fisher	12
2.5.	Cómo calcular el kernel de Fisher usando un modelo oculto de Markov	15
3.	Introducción a la biología computacional	21
3.1.	¿Qué es la bioinformática?	21
3.2.	Aplicaciones de la bioinformática	24
3.3.	Secuencias de proteínas	28
3.4.	Clasificación de secuencias	33
3.4.1.	Métodos de similitud	34
3.4.2.	Modelos probabilísticos	35
3.5.	Modelos ocultos de Markov en bioinformática	36
3.5.1.	Enfoque discriminativo	38

ÍNDICE GENERAL

4. Experimentos	41
4.1. Definición del experimento	41
4.2. Implementación de los métodos de clasificación	46
4.3. Resultados	50
5. Conclusiones y trabajos futuros	59
5.1. Conclusiones	59
5.2. Líneas futuras de trabajo	61
A. Máquinas de Vectores Soporte para clasificación	63
A.1. Descripción del problema	63
A.2. Máquinas de vectores soporte para problemas linealmente separables	65
A.3. Máquinas de vectores soporte para casos no separables	69
A.4. Generalización a problemas no lineales	72
A.5. Funciones de kernel	73
A.5.1. Funciones de kernel habituales	75
B. Modelos Ocultos de Markov	77
B.1. Modelado de fuentes de información con Modelos de Markov	77
B.2. Aplicaciones básicas de los modelos de Markov	79
B.2.1. Problema de evaluación en HMM	79
B.3. Otros parámetros de los HMM útiles para calcular el kernel de Fisher	82
C. Prestaciones del kernel de Fisher	85
Bibliografía	103
II Presupuesto	I

ÍNDICE DE FIGURAS

3.1. Evolución del número de entradas en el <i>Protein Data Base</i>	25
3.2. Estructura de un Aminoácido	30
3.3. Enlace peptídico	30
3.4. Estructura secundaria de una proteína.	32
3.5. Arquitectura de un modelo oculto de Markov de perfil. .	37
4.1. Separación de la base de datos SCOP PDB90 en los conjuntos de entrenamiento y test.	46
4.2. Arquitectura del modelo de Markov empleado en las simulaciones.	48
4.3. Curva de prestaciones para la familia 2.34.1.1 empleando una máquina lineal.	55
A.1. Máximo margen en un problema de clasificación.	66
A.2. Margen en caso no separable	70
A.3. Frontera no lineal de una SVM	74
B.1. Ejemplo de arquitectura de un modelo de Markov	78
C.1. Curva de prestaciones para la familia 1.1.1.2.	86
C.2. Curva de prestaciones para la familia 1.25.1.1.	86
C.3. Curva de prestaciones para la familia 1.25.1.2.	87
C.4. Curva de prestaciones para la familia 1.25.1.3.	87
C.5. Curva de prestaciones para la familia 1.34.1.4.	88
C.6. Curva de prestaciones para la familia 1.34.1.5.	88

ÍNDICE DE FIGURAS

C.7. Curva de prestaciones para la familia 2.1.1.1.	89
C.8. Curva de prestaciones para la familia 2.1.1.2.	89
C.9. Curva de prestaciones para la familia 2.1.1.3.	90
C.10. Curva de prestaciones para la familia 2.1.1.4.	90
C.11. Curva de prestaciones para la familia 2.1.1.5.	91
C.12. Curva de prestaciones para la familia 2.19.1.1.	91
C.13. Curva de prestaciones para la familia 2.31.1.1.	92
C.14. Curva de prestaciones para la familia 2.31.1.2.	92
C.15. Curva de prestaciones para la familia 2.34.1.1.	93
C.16. Curva de prestaciones para la familia 2.41.1.1.	93
C.17. Curva de prestaciones para la familia 2.5.1.1.	94
C.18. Curva de prestaciones para la familia 2.5.1.3.	94
C.19. Curva de prestaciones para la familia 2.8.1.2.	95
C.20. Curva de prestaciones para la familia 2.8.1.4.	95
C.21. Curva de prestaciones para la familia 3.1.1.1.	96
C.22. Curva de prestaciones para la familia 3.1.1.3.	96
C.23. Curva de prestaciones para la familia 3.1.1.5.	97
C.24. Curva de prestaciones para la familia 3.19.1.1.	97
C.25. Curva de prestaciones para la familia 3.19.1.4.	98
C.26. Curva de prestaciones para la familia 3.19.1.5.	98
C.27. Curva de prestaciones para la familia 3.25.1.1.	99
C.28. Curva de prestaciones para la familia 3.25.1.3.	99
C.29. Curva de prestaciones para la familia 3.33.1.1.	100
C.30. Curva de prestaciones para la familia 3.33.1.5.	100
C.31. Curva de prestaciones para la familia 3.50.1.7.	101
C.32. Curva de prestaciones para la familia 3.73.1.2.	101

ÍNDICE DE TABLAS

3.1. Lista de URL de distintas bases de datos bioinformáticas	26
3.2. Correspondencia entre los diferentes codones y los aminoácidos	29
3.3. Lista de aminoácidos	31
4.1. Número de secuencias en los conjuntos de entrenamiento.	44
4.2. Número de secuencias en los conjuntos de test.	45
4.3. Resultados utilizando Blast.	51
4.4. Resultados utilizando el kernel de Fisher con una máquina lineal.	53
4.5. Resultados comparados con Blast de una SVM lineal. . .	54
4.6. Resultados utilizando el kernel de Fisher con una máquina gaussiana.	57
4.7. Resultados comparados con Blast de una SVM no lineal.	58

Parte I

Proyecto Fin de Carrera

CAPÍTULO 1

Introducción

En el presente proyecto se estudiarán las bases teóricas del *kernel de Fisher* para, posteriormente, comprobar su funcionamiento con una implementación práctica del mismo en el ámbito de la bioinformática. El *kernel de Fisher* proporciona una manera sencilla y clara de unir la capacidad de los modelos de probabilidad generativos, en particular los *modelos ocultos de Markov (HMM)*, con la potencia de las *máquinas de vectores soporte (SVM)*. Una de las ventajas más evidentes de esta unión es que proporciona la capacidad de trabajar con secuencias a las SVM, aproximación que se ha intentado con métodos *ad hoc* poco generales. El *kernel de Fisher* construye un marco general para realizar esta tarea, fácilmente aplicable a cualquier modelo de probabilidad generativo.

Con la implementación práctica del *kernel de Fisher* se abordará el problema de la *búsqueda de homologías remotas en secuencias de proteínas*. Los resultados se compararán con los obtenidos usando la herramienta *BLAST (Basic Local Alignment Search Tool)*, normalmente empleada en este ámbito, para comprobar su funcionamiento.

1.1. Estado del arte

Mucha de la información que se maneja a diario está representada en forma de una secuencia, tanto temporal como lineal. Sonidos, imágenes, textos, proteínas, etc. tienen en común una representación secuencial. Para trabajar con ellas, en el ámbito del tratamiento estadístico de la

1. INTRODUCCIÓN

información, se recurre a dos tipos de métodos: aquellos basados en modelos probabilísticos y los basados en métodos de similitud. Estos últimos presentan grandes limitaciones dada su simpleza. Se basan únicamente en las diferencias que, posición a posición, muestran dos secuencias. Los modelos probabilísticos proporcionan, en general, mejores resultados. Permiten trabajar tanto con secuencias de tamaño variable como con pérdidas de fragmentos en las mismas e incluso son capaces de incorporar conocimiento previo sobre las secuencias. Con estos modelos se pueden construir funciones discriminativas que permitan diferenciar secuencias en las tareas de decisión. El problema es que estas aproximaciones pueden no funcionar correctamente si los modelos no están completamente adecuados a las secuencias con las que se trabaja. Esto puede ocurrir, simplemente, por disponer de un número limitado de muestras de las que extraer información.

Las máquinas de vectores soporte están, a día de hoy, a la cabeza de los métodos discriminativos de clasificación. Ésto se debe, principalmente, a la capacidad que tienen este tipo de máquinas de construir fronteras de decisión flexibles proporcionando, a su vez, buena generalización. Resultaría interesante emplear toda esta capacidad en problemas cuya representación sean secuencias, para aprovecharse de una mejor discriminación de la que normalmente brindan las aproximaciones basadas en modelos probabilísticos.

Desgraciadamente, las máquinas de vectores soporte no trabajan de forma directa con secuencias. Se han propuesto múltiples métodos para conseguir que las SVM puedan hacerlo. Las principales vías para conseguirlo consisten en tratar las secuencias como un vector y en extraer una serie de parámetros de la secuencias, para construir un vector de características. En el primer caso se intenta definir una métrica en relación con los símbolos de la secuencia. En muchas situaciones tal relación no existe, por ejemplo, porque los símbolos no sean más que una representación arbitraria de la información. Esta forma de proceder se revela como poco general. Además tiene la limitación de que, únicamente, puede trabajar con secuencias del mismo tamaño. La otra aproximación tiene el problema de saber cuales son las características importantes que se deben extraer de la secuencia. Por ejemplo, para la clasificación de texto, el vector de características se suele construir contabilizando las ocurrencias de una serie de palabras fijadas a priori. Lo que no se sabe es qué palabras escoger, ni con qué criterio hacerlo, por lo que esta aproximación también muestra limitaciones.

El *kernel de Fisher* presenta una forma natural de unir la capacidad discriminativa de las SVM y la potencia expresiva de los modelos de probabilísticos. Bajo la forma de una función de núcleo (*kernel*), con la que la SVM trabaja de forma natural, se define una métrica entre secuencias que permite dar una medida de lo parecidas o diferentes, de lo cerca o lo lejos que están dos secuencias.

1.2. Objetivos

Los objetivos de este proyecto pueden agruparse en tres puntos:

- Revisar las principales tareas dentro del mundo de la bioinformática, centrándose en las relacionadas con secuencias de proteínas.
- Realizar un estudio teórico sobre el kernel de Fisher.
- Implementar y validar dicho kernel en un problema de bioinformática, comparando los resultados obtenidos con una herramienta normalmente empleada para dicha aplicación. La herramienta seleccionada es el programa *Blast*.

1.3. Organización del proyecto

La memoria se organiza en en cinco capítulos, que reflejan el trabajo realizado durante el desarrollo del proyecto fin de carrera.

- Este capítulo, el primero, ha realizado una breve introducción del proyecto y de los objetivos que intenta resolver.
- El segundo capítulo introduce el *kernel de Fisher*, en su base matemática. En él se comenta como se obtiene, que ventajas presenta frente a otros métodos y de que forma puede calcularse eficientemente sobre un modelo de Markov, para emplearlo como función de kernel de una máquina de vectores soporte.
- A continuación, el tercer capítulo describe el problema sobre el que se aplicará el kernel de Fisher. Se trata de la clasificación de secuencias de proteínas. Esta tarea está enmarcada dentro del mundo de la bioinformática, al que se hace una breve introducción.

1. INTRODUCCIÓN

- El cuarto capítulo recopila los experimentos realizados con el kernel de Fisher y sus resultados. Indica, a su vez, cuales fueron los datos empleados y metodología seguida durante las simulaciones.
- Por último, el quinto capítulo recapitula todas las conclusiones obtenidas durante la realización del proyecto y los experimentos que éste comprende. También se muestran aquí otros resultados obtenidos con el kernel de Fisher, para ilustrar su funcionamiento en otras aplicaciones.
- Estos cinco capítulos están complementados por tres apéndices. En ellos se incluye una breve base teórica sobre las máquinas de vectores soporte y los modelos ocultos de Markov (Apéndices A y B respectivamente), cuyo conocimiento se ha dado por supuesto en el resto de la memoria. El último apéndice, el C, presenta una recopilación de gráficas con los resultados del kernel de Fisher para el problema estudiado en el capítulo 4.

CAPÍTULO 2

Kernel de Fisher

Los métodos probabilísticos, como pueden ser los modelos de Markov, permiten trabajar con secuencias de tamaño variable y con fragmentos desconocidos dentro de estas secuencias. Por otro lado los métodos discriminativos, como pueden ser las máquinas de vectores soporte, construyen fronteras de decisión flexibles y precisas, lo que se refleja en una buena clasificación y generalización, en muchos casos, mejor que la que proporcionan los métodos basados en modelos. Un clasificador óptimo debería unir estos dos enfoques. El *kernel de Fisher* proporciona una manera natural de realizar esta unión, estableciendo un marco general para extraer una función de kernel a partir de un modelo de probabilidad generativo.

2.1. Aprendizaje máquina

La información es aquello que sirve para incrementar el grado de conocimiento sobre un tema. Pero pueden encontrarse fuentes de información de las que no se sea capaz de extraer conocimiento alguno. Si no se es capaz de extraer algo comprensible de una fuente de información, poca utilidad se le puede dar. Para conseguir comprender esta información, se pueden emplear técnicas de tratamiento estadístico. Con estas técnicas se consiguen extraer unas relaciones sobre los datos, que sirven para entender la información que encierra la fuente que antes no era comprensible de manera directa.

Hay dos posibles enfoques dentro del tratamiento estadístico de la in-

formación: la vía analítica y la máquina. En la primera, se parte de algún tipo de modelo del sistema del que se deducen una serie de expresiones analíticas que serán de utilidad a la hora de interpretar la información que éste proporciona. En la aproximación máquina, en cambio, este modelo se desconoce, o bien porque no existe o bien porque no sea posible encontrarlo. El único conocimiento que se tiene del funcionamiento del sistema son una serie de ejemplos o experiencias previas, de las que se intenta extraer toda la información sobre el funcionamiento del mismo. De esta forma se puede construir algún tipo de máquina capaz de enfrentarse a nuevos casos desconocidos con un comportamiento lo mejor posible.

Aunque la aproximación analítica es óptima, si se tiene perfectamente modelado el sistema, puede funcionar de manera pésima cuando el modelo no es totalmente correcto. Las técnicas de aprendizaje máquina se revelan más interesantes y robustas que las analíticas pues permiten enfrentarse a problemas que son totalmente desconocidos, o sobre los que se tiene muy poca información, con una probabilidad de éxito razonable.

Todos los problemas del tratamiento estadístico de información se pueden agrupar en dos conjuntos: los problemas de decisión o clasificación y los de regresión.

2.1.1. Clasificación y regresión

En decisión, el interés se centra en saber si un determinado ejemplo o muestra contiene una información en concreto. Un ejemplo sencillo puede ser reconocer si una radiografía contiene la imagen de una fractura. La clasificación es un caso generalizado de la decisión. Se trata de una decisión múltiple entre N grupos o clases, excluyentes o no. Por sencillez se considerará sólo el caso binario, el de decisión entre dos únicas clases excluyentes. La máquina buscada será una determinada función encargada de asignar cada ejemplo o patrón de entrada a uno de estos dos conjuntos. Para elegir esta función debe emplearse algún criterio, que en general consiste en buscar el sistema que produzca el mínimo para una determinada función de coste. Por ejemplo, buscar la que produzca el mínimo número de errores ([DHS01]).

En regresión, en vez de decidir entre la existencia o no de una determinada información, lo que se quiere es estimar una variable desconocida a partir de una serie de observaciones que guardan algún tipo de relación con ella.

Aunque la clasificación y la regresión tiene características propias, ambas pueden verse como casos particulares de un problema de aproximación de funciones.

- En el caso de la regresión, resulta obvio cuál es la aproximación que se hace: aproximar la función real (en el caso de que exista) que relacione las variables de entrada con las de salida.
- Para clasificación, se aproximan las probabilidades de pertenecer o no a una determinada clase, como funciones de la muestra de entrada. Dada esta similitud muchas de las conclusiones que se saquen para clasificación pueden ser extendidas a regresión, salvando, claro está, las diferencias entre estos dos problemas. Este texto se centrará en los problemas de clasificación con el kernel de Fisher, que se desarrollará a continuación.

2.2. Máquinas de vectores soporte

Estudiadas en su origen por Vapnik ([BGV92], [CV95], [Vap95] y [Vap98]), las máquinas de vectores soporte (SVM) son aproximadores universales de funciones, que pueden emplearse tanto en problemas de clasificación como de regresión. La gran ventaja que presentan este tipo de redes, frente a las tradicionales redes neuronales (MLP o RBF), es sobre todo su buena generalización, aún en problemas con muy pocas muestras.

Las SVM son una implementación aproximada del método de minimización del coste estructural. Este principio está basado en el hecho de que la tasa de error de un clasificador, sobre la distribución real de las muestras, está acotada por el sumatorio de los errores de entrenamiento y un término que depende de la *dimensión Vapnik-Chervonenkis (VC)*¹. Una máquina de vectores soporte minimiza el primer término, manteniendo el segundo lo más ajustado posible. De acuerdo con esto, las SVM pueden proporcionar una muy buena generalización en los problemas de clasificación de patrones, a pesar del hecho de que no incorpora ningún tipo de información adicional del problema, aparte de los datos del conjunto

¹Esta dimensión está relacionada con la capacidad expresiva de la máquina. Cuanto mayor sea, de más formas diferentes podrá separar un conjunto de N muestras. El problema al incrementar la dimensión VC es que con tal capacidad expresiva puede que la máquina sobreajuste de manera excesiva el conjunto de entrenamiento.

de entrenamiento. Esta característica hace especialmente interesantes a este tipo de máquinas.

Para construir la función de clasificación, se parte de un conjunto de muestras, \mathbf{X}_i , cada una de las cuales lleva asociada una etiqueta (información sobre a qué clase pertenece), $y_i, y_i \in \{\pm 1\}$. De entre todas estas muestras se extraen una serie de vectores conocidos como *vectores soporte*, los únicos que se necesitan de entre todos los disponibles para definir la frontera de decisión. Los vectores soporte son las muestras más agresivas para la búsqueda de la frontera, aquellas que más cerca se encuentran de ésta. La máquina de vectores soporte se construye como una combinación lineal de los núcleos (*kernel*) sobre estos vectores soporte y la muestra a clasificar. Estos kernel proporcionan a las máquinas de vectores soporte la capacidad de enfrentarse perfectamente a los problemas no lineales. Resumiendo, la SVM asignará a una muestra \mathbf{X} , la clase \hat{y} siguiendo la siguiente expresión:

$$\hat{y} = \text{sgn} \left(\sum_i^{N_S} y_i \alpha_i K(\mathbf{X}_i, \mathbf{X}) \right) \quad (2.1)$$

N_S indica el número de vectores soporte en el conjunto de entrenamiento. La suma tiene en cuenta dos términos:

- la importancia del vector soporte, \mathbf{X}_i , con respecto a los demás (α_i),
- y la similitud de este vector soporte con la muestra a clasificar ($K(\mathbf{X}_i, \mathbf{X})$).

Los pesos α_i , y en menor medida la función de kernel, forman los parámetros libres del sistema, teniendo gran importancia la manera en las que debemos buscarlos. Como ya se ha mencionado, el resultado final variará en función del conjunto de muestras con el que trabajemos, al ser la propia salida una combinación lineal de alguna de las muestras de este conjunto. A la hora de utilizar las SVM es necesario elegir una función de kernel apropiada para el problema que se quiera resolver. La función de kernel representa la métrica usada entre las muestras, la forma de indicar lo cerca o lejos que están dos puntos del espacio de entrada. El caso más sencillo es la utilización del producto interno en el espacio de trabajo. Esto creará un clasificador lineal, limitando los problemas que se pueden resolver a aquellos que son linealmente separables. Otros posibles kernel, que permiten trabajar con fronteras no lineales, pueden

ser el polinómico y el gaussiano. Escoger un kernel apropiado para cada problema en particular no es una tarea inmediata, en la que se sigue investigando. Una posible solución es emplear el kernel conocido como *kernel de Fisher* ([JH98]).

2.3. Motivación del Kernel de Fisher

Los modelos de probabilidad generativos son capaces de trabajar con secuencias de tamaño variable y afrontar pérdidas de fragmentos en las mismas. Pueden emplearse incluso para clasificación, construyendo funciones de discriminación. Estas funciones serán óptimas cuando los modelos estén perfectamente ajustados a las familias o clases de secuencias con las que se trate. En general no se tendrá este perfecto conocimiento del modelo, por ejemplo, por trabajar con conjuntos limitados de muestras.

Las máquinas de vectores soporte están demostrando su valía para multitud de aplicaciones, con unos resultados no alcanzados anteriormente por otros sistemas de clasificación. Dada la generalidad con la que están contruidos estos métodos, puede aplicarse su capacidad a cualquier problema en el que esté definido un *kernel adecuado*. Lo que no es tan sencillo es cómo encontrar este *kernel adecuado* para un problema en particular. El principal problema a la hora de encontrar un kernel es que resulta necesario tener algún tipo de métrica sobre los datos tratados, alguna forma de decir lo cerca o lejos que están dos muestras, en el fondo, lo parecidas que son. En el caso de las secuencias, dicha métrica no está definida, lo que impide emplear las SVM con este tipo de datos. Otras aproximaciones han intentado tratar la secuencia como un vector o extraer de ella una serie de parámetros pero no han obtenido buenos resultados. En primer lugar los símbolos que forman la secuencia pueden tener una representación arbitraria (letras, índices, etc) sobre la cual no se puede construir una métrica. Si se aborda la extracción de parámetros, el problema está en saber cuales son los necesarios, cosa que es, a priori, desconocida.

El kernel de Fisher proporciona un marco general para unir los dos enfoques anteriores, los modelos de probabilidad generativos con las máquinas de vectores soporte. El objetivo de esta unión es compensar las carencias de cada uno de los enfoques, sacando partido de sus ventajas individuales. El kernel de Fisher proporciona una función de núcleo y por tanto un tipo de métrica, sobre un conjunto de secuencias, lo que

habilita a las máquinas de vectores soporte para trabajar con este tipo de datos. Por otro lado emplea la capacidad discriminativa que demuestran las SVM en los problemas en los que la utilización únicamente de los modelos de probabilidad no proporciona resultados aceptables.

Por ésto, el kernel de Fisher aparece como un método muy interesante frente a las alternativas antes comentadas, para trabajar con secuencias y máquinas de vectores soporte, cuando la calidad de los métodos basados en modelos es insuficiente.

2.4. La función de kernel de Fisher

El objetivo del kernel de Fisher es extraer (derivar) una función de kernel de un modelo de probabilidad generativo. Ya se ha hablado del interés de este kernel, pero nada se ha dicho de cómo conseguirlo. La función debe reflejar, de alguna manera, una métrica o relación entre las muestras con las que se trabaje, sean éstas del tipo que sea. La medida $K(X, Y)^2$ indicará lo cerca o lo lejos que están las muestras X e Y según esta métrica.

Una función de kernel puede verse, de manera general, como un producto interno en un espacio de características, sobre el que han sido proyectadas las muestras de entrada (ver apéndice A.5):

$$K(X, Y) = \Phi(X)^T \Sigma \Phi(Y) \quad (2.2)$$

La forma en la que se defina el producto interno en este espacio, llevará asociada una métrica en el mismo. El kernel de Fisher extrae esta métrica de un modelo de probabilidad generativo³.

Un modelo de probabilidad generativo proporciona una medida de la verosimilitud de que una muestra haya sido o no generada por él. Dicha verosimilitud se utiliza, en una aproximación meramente discriminativa, para buscar cuál es el modelo que proporciona la mayor verosimilitud de entre los disponibles. Con esta información se puede construir un clasificador de tipo *MAP* (máximo a posteriori). Al contrario que en esta

²Nótese que se ha cambiado el vector \mathbf{x} por un dato cualquiera X . Este no tiene porqué tener un representación vectorial, pudiendo ser, para el caso que aquí se tratará, una secuencia de tamaño variable.

³Cuando el objetivo del kernel sea la clasificación, el modelo debe incluir la clase como una variable latente.

otra aproximación, lo que interesa ahora, para buscar una métrica entre muestras, es encontrar las diferencias en el proceso de generación de muestras. Lo lejos o cerca que están del modelo (verosimilitud) no proporciona información sobre si se parecen entre sí. Dos valores de verosimilitud parecidos pueden corresponder a dos secuencias radicalmente distintas.

Una forma de capturar el proceso de generación es emplear el gradiente del logaritmo de la verosimilitud, con respecto a los parámetros del modelo. Cada una de las derivadas parciales del gradiente indicará cómo afecta un parámetro en particular al cálculo de la verosimilitud para una muestra. El espacio del gradiente tiene la ventaja de que además preserva todas las suposiciones que contiene el modelo sobre el proceso de generación de muestras.

Considérense todos los posibles modelos, controlados por el vector de parámetros θ , del tipo:

$$P(X|\theta); \quad \theta \in \Theta \quad (2.3)$$

Esta clase de modelos de probabilidad definen una variedad de Riemann, M_Θ , con una métrica local dada por la matriz de información de Fisher, F ([Ama98]):

$$F = E_X \{ \mathbf{U}_X \mathbf{U}_X^T \} \quad (2.4)$$

La esperanza se realiza sobre los vectores \mathbf{U}_X , conocidos como puntuaciones de Fisher:

$$\mathbf{U}_X = \nabla_\theta \log P(X|\theta) \quad (2.5)$$

La métrica local M_Θ define una distancia entre el modelo actual, $P(X|\theta)$ y un modelo cercano, $P(X|\theta + \delta)$. Esta distancia vendrá dada por $D(\theta, \theta + \delta) = 1/2\delta^T F \delta$, que también aproxima la distancia *Kullback-Leibler* o divergencia asimétrica ([KL51]) entre dos modelos para un δ suficientemente pequeño.

La puntuación de Fisher, $\mathbf{U}_X = \nabla_\theta \log P(X|\theta)$ proyecta una muestra X en un vector que es un punto del espacio del gradiente de la variedad M_Θ . A esto se le conoce como *proyección de la puntuación de Fisher*. \mathbf{U}_X también puede usarse para definir la dirección de máxima pendiente en la función $\log P(X|\theta)$ para una muestra X en la variedad M_Θ . Este gradiente se conoce como el *gradiente natural* ([Ama98]) y se obtiene sustituyendo ϕ_X por $F^{-1}\mathbf{U}_X$. La función de proyección $X \mapsto \phi_X$ es la

2. KERNEL DE FISHER

proyección natural de las muestras en los vectores de características. El producto interno en el espacio de ϕ relativo a la métrica local de Riemann define el kernel natural, que puede expresarse en función de los vectores de puntuación de Fisher de la siguiente forma:

$$K(X_i, X_j) \propto \phi_{X_i}^T F \phi_{X_j} = (F^{-1} \mathbf{U}_{X_i})^T F (F^{-1} \mathbf{U}_{X_j}) = \mathbf{U}_{X_i}^T F^{-1} \mathbf{U}_{X_j} \quad (2.6)$$

Éste es el llamado kernel de Fisher. Su nombre se debe al importante papel que juegan las puntuaciones de Fisher en su construcción.

Las propiedades de este kernel se pueden agrupar en el siguiente teorema ([JH98]):

Teorema:

Para todo modelo $P(X|\theta)$ con parámetros θ , el kernel de Fisher

$$K(X, Y) = \mathbf{U}_X^T F^{-1} \mathbf{U}_Y$$

donde $\mathbf{U}_X = \nabla_{\theta} \log P(X|\theta)$, tiene las siguientes propiedades:

- *Es una función de kernel válida.*
- *Es invariable sobre cualquier transformación invertible y diferenciable de los parámetros.*
- *Un clasificador empleando el kernel de Fisher derivado de un modelo que incluya la clase como una variable latente es, asintóticamente por lo menos, tan bueno como un clasificador MAP (máximo a posteriori) basado en dicho modelo.*

La primera propiedad es evidente en cuanto la matriz de información de Fisher, F , es definida positiva. La segunda propiedad sigue el hecho de que el kernel ha sido derivado teniendo en cuenta sólo la variedad M_{Θ} , la cual es intrínseca a la clase de los modelos probabilísticos y es independiente de la parametrización. La tercera propiedad puede extraerse de las bases de la derivación discriminativa del kernel ([JH98]).

Un kernel, como puede ser éste, sólo nos brinda un medio para comparar secuencias entre sí. Para afrontar una tarea de clasificación se debe emplear algún clasificador de los basados en kernel. Aquí se discutirá la utilización de este kernel con las máquinas de vectores soporte. Las versiones cuadráticas, cúbicas, etc, del kernel de Fisher también se pueden manejar para un caso no lineal:

$$\hat{K}(X, Y) = (1 + K(X, Y))^p \quad (2.7)$$

$$(2.8)$$

Incluso se puede trabajar con kernel gaussianos recurriendo a una medida de distancia natural entre vectores ([JDH98]). Ésta está dada por:

$$D^2(X, Y) = \frac{1}{2}(\mathbf{U}_X - \mathbf{U}_Y)^T F^{-1}(\mathbf{U}_X - \mathbf{U}_Y) \quad (2.9)$$

Lo que permite emplear el siguiente kernel gaussiano:

$$K(X, Y) = e^{-D^2(X, Y)} \quad (2.10)$$

2.5. Cómo calcular el kernel de Fisher usando un modelo oculto de Markov

Como ya se ha visto, la aplicación del kernel de Fisher parte de la extracción de una serie de parámetros relevantes. Éstos nos darán información acerca del proceso de generación de una determinada secuencia por el modelo. A continuación se tratará el caso en el que dicho modelo sea un modelo oculto de Markov discreto ([Rab89], apéndice B). Los parámetros que definen uno de estos modelos, λ , son los siguientes:

- La probabilidad de saltar a un estado, j , dado que se parte del estado i , a_{ij} .
- La probabilidad en emitir un símbolo x dado que se encuentra en el estado j , $b_j(x)$.
- La probabilidad de estar situado en el estado j al inicio de la secuencia, π_j .

Aunque éstos son todos los parámetros que controlan el kernel de Fisher, sólo se van a considerar los gradientes con respecto a las probabilidades de emisión. Se usan estos parámetros ya que son las que más importancia tienen en el proceso de generación, relegando a los demás a un segundo plano. Dado que estas probabilidades no son independientes

2. KERNEL DE FISHER

entre sí, ya que $\sum_x b_j(x) = 1$, es conveniente reformular estas probabilidades de la siguiente forma:

$$b_j(x) = P(x|q_j) = \frac{P(x, q_j)}{P(q_j)} = \frac{P(x, q_j)}{\sum_{x'} P(x', q_j)} \quad (2.11)$$

donde $P(x, q_j)$ es la probabilidad conjunta de emitir el residuo x y encontrarse en el estado q_j . Asumiendo que los valores de $P(x, q_j)$ se ajustan de tal forma que $\sum_{x'} P(x', q_j) = 1$, se obtiene que la probabilidad conjunta y la condicionada son iguales, $b_j(x) = P(x|q_j) = P(x, q_j)$. La puntuación de Fisher para una secuencia será el gradiente de su verosimilitud con respecto a estas probabilidades conjuntas.

$$\mathbf{U}_X = \nabla_{P(x,q)} \log P(X|\lambda) \quad (2.12)$$

La verosimilitud de una secuencia⁴ X , $X = x_1, x_2, \dots, x_l$, cualquiera, dado el modelo λ , es

$$\begin{aligned} P(X|\lambda) &= \sum_{Q_j} \prod_i P(x_i|q_i, \lambda) P(q_i|q_{i-1}, \lambda) \\ &= \sum_{Q_j} \prod_i b_{q_i}(x_i) a_{q_{i-1}q_i} \end{aligned} \quad (2.13)$$

donde Q_j , $Q_j = Q_1, \dots, Q_n$ representa todos los posibles caminos que puede seguir la secuencia en su proceso de generación dentro del modelo. En cada uno de estos caminos, la probabilidad de que se genere la secuencia de observaciones X en particular y no otra cualquiera se calcula como la probabilidad de haber estado en todos los estados de la secuencia Q_j , y haber emitido en cada uno de ellos el símbolo x_i , $\prod_i b_{q_i}(x_i) a_{q_{i-1}q_i}$.

El vector \mathbf{U}_X tendrá tantas componentes como parámetros del modelo se consideren en el gradiente. Tomando las probabilidades de emisión, el tamaño del vector \mathbf{U}_X será igual al número de estados multiplicado por el número de símbolos posibles (tamaño de la matriz B). Cada una de las

⁴A partir de ahora a la muestras con las que trabajemos se las denominará secuencias. Los modelos de Markov no trabajan con vectores sino con secuencias de símbolos o secuencias de vectores. En ningún momento se ha puesto ninguna restricción al tamaño de la secuencia ni a su contenido.

2.5. Cómo calcular el kernel de Fisher usando un modelo oculto de Markov

dimensiones de \mathbf{U}_X será una derivada parcial con respecto a un parámetro en particular:

$$\mathbf{U}_X(\tilde{x}, \tilde{q}) = \frac{\partial}{\partial P(\tilde{x}, \tilde{q})} \log P(X|\lambda) \quad (2.14)$$

Se comienza a derivar por el logaritmo:

$$\frac{\partial}{\partial P(\tilde{x}, \tilde{q})} \log P(X|\lambda) = \frac{1}{P(X|\lambda)} \frac{\partial}{\partial P(\tilde{x}, \tilde{q})} P(X|\lambda) \quad (2.15)$$

A continuación se puede sustituir la verosimilitud dada por el modelo en la expresión (2.13):

$$\begin{aligned} \frac{\partial}{\partial P(\tilde{x}, \tilde{q})} \log P(X|\lambda) &= \\ &= \frac{1}{P(X|\lambda)} \frac{\partial}{\partial P(\tilde{x}, \tilde{q})} \sum_{Q_j} \prod_i b_{q_i}(x_i) a_{q_{i-1}q_i} \end{aligned} \quad (2.16)$$

$$= \frac{1}{P(X|\lambda)} \sum_{Q_j} \frac{\partial}{\partial P(\tilde{x}, \tilde{q})} \prod_i b_{q_i}(x_i) a_{q_{i-1}q_i} \quad (2.17)$$

En este punto, para simplificar, se puede abordar el cálculo de la derivada parcial de forma independiente. La parcial del productorio quedará como sigue:

$$\begin{aligned} \frac{\partial}{\partial P(\tilde{x}, \tilde{q})} \prod_i b_{q_i}(x_i) a_{q_{i-1}q_i} &= \\ &= \sum_k \left[\frac{\partial}{\partial P(\tilde{x}, \tilde{q})} b_{q_k}(x_k) \right] a_{q_{k-1}q_k} \prod_{i \neq k} b_{q_i}(x_i) a_{q_{i-1}q_i} \end{aligned} \quad (2.18)$$

Ahora sustituyendo la probabilidad de emisión condicionada por la conjunta se obtiene:

$$\begin{aligned}
 & \frac{\partial}{\partial P(\tilde{x}, \tilde{q})} \prod_i b_{q_i}(x_i) a_{q_{i-1}q_i} = \\
 & = \sum_k \left[\frac{\partial}{\partial P(\tilde{x}, \tilde{q})} \left(\frac{P(x_k, q_k)}{\sum_{x'} P(x', q_k)} \right) \right] a_{q_{k-1}q_k} \prod_{i \neq k} b_{q_i}(x_i) a_{q_{i-1}q_i} \quad (2.19)
 \end{aligned}$$

$$\begin{aligned}
 & = \sum_k \left[\frac{\delta_{q_k, \tilde{q}} \delta_{x_k, \tilde{x}}}{\sum_{x'} P(x', q_k)} - \frac{P(x, q_k) \sum_{x'} \delta_{q_k, \tilde{q}} \delta_{x', \tilde{x}}}{(\sum_{x'} P(x', q_k))^2} \right] a_{q_{k-1}q_k} \prod_{i \neq k} b_{q_i}(x_i) a_{q_{i-1}q_i} \\
 & \quad (2.20)
 \end{aligned}$$

Para continuar debe tenerse en cuenta, como se indicó anteriormente, que $P(x, q_j) = b_j(x)$ y que $\sum_{x'} P(x', q_k) = 1$. También se puede simplificar $\sum_{x'} \delta_{q_k, \tilde{q}} \delta_{x', \tilde{x}}$ por $\delta_{q_k, \tilde{q}}$. Introduciendo estos cambios se obtiene:

$$\begin{aligned}
 & \frac{\partial}{\partial P(\tilde{x}, \tilde{q})} \prod_i b_{q_i}(x_i) a_{q_{i-1}q_i} = \\
 & = \sum_k [\delta_{q_k, \tilde{q}} \delta_{x_k, \tilde{x}} - b_{q_k}(x_k) \delta_{q_k, \tilde{q}}] a_{q_{k-1}q_k} \prod_{i \neq k} b_{q_i}(x_i) a_{q_{i-1}q_i} \quad (2.21)
 \end{aligned}$$

$$\begin{aligned}
 & = \sum_k \left[\frac{\delta_{q_k, \tilde{q}} \delta_{x_k, \tilde{x}}}{b_{q_k}(x_k)} - \delta_{q_k, \tilde{q}} \right] b_{q_k}(x_k) a_{q_{k-1}q_k} \prod_{i \neq k} b_{q_i}(x_i) a_{q_{i-1}q_i} \quad (2.22)
 \end{aligned}$$

$$\begin{aligned}
 & = \sum_k \left[\frac{\delta_{q_k, \tilde{q}} \delta_{x_k, \tilde{x}}}{b_{q_k}(x_k)} - \delta_{q_k, \tilde{q}} \right] \prod_i b_{q_i}(x_i) a_{q_{i-1}q_i} \quad (2.23)
 \end{aligned}$$

$$\begin{aligned}
 & = \sum_k \left[\frac{\delta_{q_k, \tilde{q}} \delta_{x_k, \tilde{x}}}{b_{\tilde{q}}(\tilde{x})} - \delta_{q_k, \tilde{q}} \right] \prod_i b_{q_i}(x_i) a_{q_{i-1}q_i} \quad (2.24)
 \end{aligned}$$

Ahora introduciendo este resultado en la expresión (2.17), cambiando el orden de los sumatorios y simplificando se obtiene lo siguiente:

2.5. Cómo calcular el kernel de Fisher usando un modelo oculto de Markov

$$\mathbf{U}_X = \frac{1}{P(X|\lambda)} \sum_{Q_j} \sum_k \left[\frac{\delta_{q_k, \tilde{q}} \delta_{x_k, \tilde{x}}}{b_{\tilde{q}}(\tilde{x})} - \delta_{q_k, \tilde{q}} \right] \prod_i b_{q_i}(x_i) a_{q_{i-1} q_i} \quad (2.25)$$

$$\begin{aligned} &= \frac{1}{b_{\tilde{q}}(\tilde{x})} \sum_k \sum_{Q_j} \frac{\delta_{q_k, \tilde{q}} \delta_{x_k, \tilde{x}}}{P(X|\lambda)} \prod_i b_{q_i}(x_i) a_{q_{i-1} q_i} \\ &\quad - \sum_k \sum_{Q_j} \frac{\delta_{q_k, \tilde{q}}}{P(X|\lambda)} \prod_i b_{q_i}(x_i) a_{q_{i-1} q_i} \end{aligned} \quad (2.26)$$

$$\begin{aligned} &= \frac{1}{b_{\tilde{q}}(\tilde{x})} \sum_k \sum_{Q_j} \delta_{q_k, \tilde{q}} \delta_{x_k, \tilde{x}} \frac{P(Q_j, X|\lambda)}{P(X|\lambda)} \\ &\quad - \sum_k \sum_{Q_j} \delta_{q_k, \tilde{q}} \frac{P(Q_j, X|\lambda)}{P(X|\lambda)} \end{aligned} \quad (2.27)$$

$$\begin{aligned} &= \frac{1}{b_{\tilde{q}}(\tilde{x})} \sum_k \sum_{Q_j} \delta_{q_k, \tilde{q}} \delta_{x_k, \tilde{x}} P(Q_j|X, \lambda) \\ &\quad - \sum_k \sum_{Q_j} \delta_{q_k, \tilde{q}} P(Q_j|X, \lambda) \end{aligned} \quad (2.28)$$

$$= \frac{1}{b_{\tilde{q}}(\tilde{x})} \sum_k E\{\delta_{q_k, \tilde{q}} \delta_{x_k, \tilde{x}} | X, \lambda\} - \sum_k E\{\delta_{q_k, \tilde{q}} | X, \lambda\} \quad (2.29)$$

$$= \frac{\xi(\tilde{x}, \tilde{q})}{b_{\tilde{q}}(\tilde{x})} - \xi(\tilde{q}) \quad (2.30)$$

$P(Q_j, X|\lambda)$ es la probabilidad de tomar un camino Q_j y tener la secuencia X con un determinado modelo. $\sum_j P(Q_j, X|\lambda)$ será la verosimilitud de la secuencia X para el modelo λ , $P(X|\lambda)$. El resultado de la puntuación de Fisher está expresado en función de la variable ξ .

$$\mathbf{U}_X = \frac{\xi(\tilde{x}, \tilde{q})}{b_{\tilde{q}}(\tilde{x})} - \xi(\tilde{q}) \quad (2.31)$$

Esta variable es igual a:

$$\xi(\tilde{x}, \tilde{q}) = \sum_k E\{\delta_{q_k, \tilde{q}} \delta_{x_k, \tilde{x}} | X, \lambda\} \quad (2.32)$$

Las variables ξ son estadísticos suficientes para las probabilidades de emisión. Se pueden obtener de forma directa y eficiente del algoritmo

forward-backward (sección B.2.1). El parámetro $\gamma_{x_k}(q_j)$ de este algoritmo es la probabilidad de estar en el estado q_j en el instante x_k . Así puede calcularse ξ , acumulando esta variable cuando $x_k = \tilde{x}$:

$$\xi(\tilde{x}, \tilde{q}) = \sum_{k / x_k = \tilde{x}} \gamma_{x_k}(\tilde{q}) \quad (2.33)$$

$\xi(\tilde{x}, \tilde{q})$ representa el número de veces que se ha estado en el estado \tilde{q} , durante el proceso de generación de la secuencia, emitiendo el símbolo \tilde{x} . $\xi(\tilde{q})$ es el número de veces que se ha estado en el estado \tilde{q} durante el proceso de generación de la secuencia.

$$\xi(\tilde{q}) = \sum_k \gamma_{x_k}(\tilde{q}) = \sum_{\tilde{x}} \xi(\tilde{x}, \tilde{q}) \quad (2.34)$$

El parámetro $\gamma_{x_k}(q_j)$ se obtiene de las variables α y β del algoritmo *forward-backward*). Se puede expresar en función de estas dos variables de la siguiente forma:

$$\gamma_{x_k}(q_j) = \frac{\alpha_{x_k}(q_j)\beta_{x_k}(q_j)}{P(X|\lambda)} = \frac{\alpha_{x_k}(q_j)\beta_{x_k}(q_j)}{\sum_{j=1}^N \alpha_{x_k}(q_j)\beta_{x_k}(q_j)} \quad (2.35)$$

CAPÍTULO 3

Introducción a la biología computacional

La bioinformática es una disciplina en continuo crecimiento, maneja grandes cantidades de datos que resultan difíciles de interpretar por un humano. En este contexto, las técnicas de tratamiento estadístico de la información y de aprendizaje máquina surgen como una solución a esta limitación. Dentro de las tareas que aborda la bioinformática se encuentra la búsqueda de homologías entre las secuencias de aminoácidos que forman las proteínas. Los métodos actuales para realizar esta búsqueda presenta principalmente dos enfoques: utilizar alineamientos de secuencias en función de las similitudes de la misma y construir modelos probabilísticos para utilizar una función discriminativa. Una alternativa a estos métodos es la utilización del kernel de Fisher sobre los modelos probabilísticos para aumentar la capacidad discriminativa del sistema.

3.1. ¿Qué es la bioinformática?

Un laboratorio es capaz de generar 100 Gigabytes de información en un solo día, que deberá ser analizada, clasificada y comparada con otras fuentes de información. Multitud de bases de datos a lo largo de todo el mundo siguen creciendo, alcanzando unos volúmenes inmensos. Por ejemplo, en agosto de 2002, la base de datos *Genebank* de ácidos nucleicos contenía más de 22.000 millones de bases en más de 19 millones

3. INTRODUCCIÓN A LA BIOLOGÍA COMPUTACIONAL

de secuencias. Esta misma base de datos, tan solo dos años antes, en agosto de 2000 contenía únicamente 8 millones de secuencias. Del mismo modo, el *SWISS-PROT*, base que almacena secuencias de proteínas, contenía 88.000 secuencias en el año 2000. En octubre de 2002 alcanzó las 115.000, entre las que sumaban más de 42 millones de aminoácidos. Existen multitud de bases de estos tipos, y todas mantienen grandes cantidades de datos, lo que hace difícil trabajar con ellas.

Manejando estas cantidades de datos, resulta imperativo tener herramientas que ayuden a trabajar con ellos, tanto en las tareas sencillas de almacenamiento y búsqueda como en las más complicadas, que involucren la comprensión del significado de los datos con los que se está trabajando.

Todas estas actividades están dentro del ámbito de la *bioinformática*, un campo interdisciplinario situado en la intersección de la biología con las ciencias de la información.

La bioinformática estudia la transmisión de información desde los genes a las estructuras moleculares. Para ello, desarrolla herramientas encaminadas a comprender esta transmisión, entender su función bioquímica, su conducta biológica y, finalmente, averiguar qué influencia tienen en las enfermedades y en la salud. Los objetivos que la bioinformática persigue, pueden agruparse en los siguientes campos:

- En primer lugar, la bioinformática se encarga de organizar los datos de tal manera que los investigadores puedan acceder a toda la información existente de forma sencilla, productiva y, además, tengan la oportunidad de agregar nuevos datos que sean de utilidad. Existen multitud de formatos de representación de secuencias de proteínas o de descripción de macromoléculas. Las bases de datos en Internet albergan esta información, proporcionando acceso a la misma de manera sencilla. Entre estas bases de datos pueden destacarse las siguientes: *SWISS-PROT*, *PIR-International*, *OWL*, *NRDB*, *PROSITE*, *PRINTS*, *Pfam*, *Protein Data Bank (PDB)*, *Nucleic Acids Database (NDB)*, *HIV Protease Database*, *SCOP*, *GenBank*, *EMBL*, *DDBJ*, etc. Uno de los problemas con los que se enfrenta actualmente la bioinformática es precisamente, aunque parezca extraño, esta gran diversidad de representación, lo que dificulta la interacción entre las bases de datos. En la actualidad, varios proyectos intentan estandarizar la representación, siendo aquellos que emplean XML para describir los contenidos los que están adquiriendo más fuerza.

- El segundo objetivo es desarrollar herramientas y recursos que ayuden al análisis de todos estos datos para poder extraer algún tipo de información. Así van surgiendo multitud de algoritmos y aplicaciones que ayudan en cada uno de los problemas planteados en el trabajo de laboratorio diario. Ejemplos de estas tareas pueden ser la secuenciación de un genoma, la comparación de proteínas o la predicción del comportamiento de un determinado medicamento en un organismo.
- El tercer propósito consiste en unir los dos anteriores y conseguir acercar al trabajo diario este tipo de herramientas. De estas herramientas debe extraerse información biológica interesante que pueda emplearse para aumentar el grado de conocimiento sobre los problemas tratados. Este apartado es más afín al mundo de la biología que al del tratamiento estadístico de la información, puesto que es aquella ciencia la encargada de interpretar los resultados obtenidos.

Al hablar de bioinformática, se hace referencia a las aplicaciones de la teoría de la información a la biología molecular. Éstas comprenden desde la información almacenada en el ADN de los organismos hasta la interacción de las moléculas entre sí (por ejemplo la interacción entre determinadas proteínas y sus ligandos). Muchas veces se introducen erróneamente aplicaciones más cercanas a la biomedicina o a otros campos similares.

Definición de bioinformática

Hasta ahora simplemente se han enumerado una serie de funciones que cumple la bioinformática. Sin embargo, resulta difícil dar una definición suficientemente concisa de lo que representa esta palabra. Una de las que mejor se ajusta, y que a la vez abarca todo lo comentado hasta ahora, puede encontrarse en [LGG]:

“La bioinformática conceptualiza la biología en términos de moléculas (en el sentido de la física química) y aplica técnicas informáticas para entender y organizar toda información asociada a esas moléculas, a gran escala. A pequeña escala, la bioinformática es un sistema de administración de información para la biología molecular que tiene muchas aplicaciones prácticas”.¹

¹Definición enviada al diccionario *Oxford* de la lengua inglesa

Esta *aplicación de técnicas informáticas* a la que hace referencia, abarca muchos campos dentro de lo que es la biología. Pueden encontrarse aplicaciones desde los niveles moleculares dentro de las células, hasta la descripción de los contenidos para el correcto y fructífero intercambio de información por parte de la comunidad científica. La bioinformática se revela como un campo muy extenso que hace uso constante de muchas de las actuales ramas de investigación en las tecnologías de la información.

3.2. Aplicaciones de la bioinformática

El flujo de información que estudia la bioinformática comienza en el ADN de los seres vivos. Las largas secuencias de ADN, compuestas por las cuatro posibles bases (*adenina (A)*, *citocina (C)*, *guanina (G)* y *timina (T)*) se encuentran en todas las células de un ser vivo, agrupadas en cromosomas. Un cromosoma es una larga secuencia de ADN, aunque no toda ella contendrá información útil. Una de las principales tareas, una vez secuenciado el genoma de un organismo (secuenciados todos sus cromosomas), consiste en identificar cuales son las partes que tienen información y cuales no. Las partes que contienen información, por lo general, codificarán proteínas. Cada proteína es una secuencia de 20 posibles aminoácidos, que al plegarse constituyen una macromolécula cuya forma le confiere unas características especiales, capacitándola para una u otra función. Estas macromoléculas interactúan entre sí y realizan funciones en el organismo.

Cada tipo de datos en particular, necesita una representación adecuada. Con el ADN y las proteínas suelen emplearse secuencias de bases (4 símbolos) o de aminoácidos (20 símbolos) respectivamente. Para las proteínas y otras macromoléculas es posible trabajar con la información de su estructura tridimensional. El procedimiento habitual consiste en almacenar la posición en los tres ejes de todos los átomos que componen la molécula. A parte de estas representaciones básicas, según la tarea a resolver, pueden encontrarse otras muchas alternativas.

Como se puede ver, los tipos de datos usados son muy variados. Cada uno de ellos lleva asociado una información diferente, aunque en muchos casos pueden complementarse con otras fuentes. Comúnmente se emplean una serie herramientas bien conocidas para trabajar con cada una de las fuentes típicas. Otras muchas técnicas están siendo probadas como mejoras a los métodos anteriores, o con enfoques totalmente nuevos,

intentando dar soluciones a todas las necesidades que se van planteando.

Aún con toda esta diversidad de métodos, un concepto común subyace bajo ellos. La mayor parte de los datos utilizados pueden agruparse de forma significativa, basándose en similitudes biológicas. Los genes pueden diferenciarse según cumplan una determinada función (por ejemplo las acciones enzimáticas), de acuerdo a la función metabólica a la que pertenecen o atendiendo a otras consideraciones. Yendo más lejos, proteínas diferentes normalmente poseen secuencias comparables. Los organismos suelen tener múltiples copias de un gen debido a la duplicación y diferentes especies tienen proteínas similares o equivalentes que se heredaron cuando divergieron durante la evolución. A un nivel estructural, se cree que existe un número finito de posibles estructuras terciarias de las proteínas (ver sección 3.3) e incluso secuencias muy diferentes pueden adoptar la misma estructura terciaria (o similares). Esto hace que, aunque el número de entradas en el Protein Database² (PDB) siga creciendo a gran velocidad, la frecuencia con la que se descubren nuevos pliegues haya disminuido (figura 3.1).

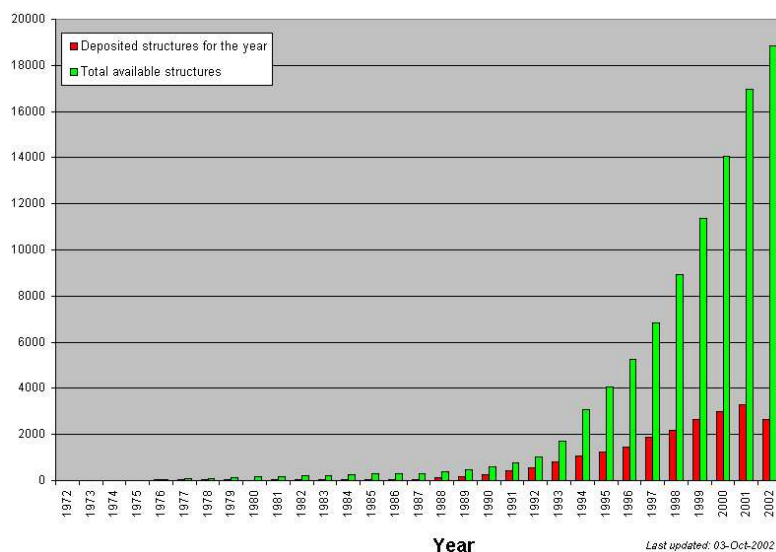


Figura 3.1: Evolución del número de entradas en el *Protein Data Base*

Las tareas más típicas a las que debe hacer frente la bioinformática en función de los tipos de datos a analizar son:

²Base de datos que contiene secuencias de proteínas.

3. INTRODUCCIÓN A LA BIOLOGÍA COMPUTACIONAL

Base de datos	URL
Secuencias de aminoácidos	
GenBank	http://www.ncbi.nlm.nih.gov/Genbank
EMBL	http://www.ebi.ac.uk/embl
DDBJ	http://www.ddbj.nig.ac.jp/E-mail/homology.html
Secuencias de proteínas (Primarias)	
SWISS-PROT	http://us.expasy.org/sprot
PIR-International	http://mips.gsf.de/proj/protseqdb
Secuencias de proteínas (Compuestas)	
OWL	http://bioinf.man.ac.uk/dbbrowser/OWL
NRDB	http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=Protein
Secuencias de proteínas (Secundarias)	
PROSITE	http://www.expasy.org/prosite
PRINTS	http://www.bioinf.man.ac.uk/dbbrowser/PRINTS
Pfam	http://www.sanger.ac.uk/Pfam
Estructuras de macromoléculas	
Protein Data Bank (PDB)	http://www.rcsb.org/pdb
Nucleic Acids Database (NDB)	http://ndbserver.rutgers.edu
HIV Protease Database	http://mcl1.ncifcrf.gov/hivdb
ReLiBase	http://www.ccdc.cam.ac.uk/prods/relibase
PDBsum	http://www.biochem.ucl.ac.uk/bsm/pdbsum
CATH	http://www.biochem.ucl.ac.uk/bsm/cath_new
SCOP	http://scop.mrc-lmb.cam.ac.uk/scop
FSSP	http://www.ebi.ac.uk/dali/fssp
Secuencias de genomas	
Entrez genomes	http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=Genome
GeneCensus	http://bioinfo.mbb.yale.edu/genome
COGs	http://www.ncbi.nlm.nih.gov/COG
Bases de datos integradas	
InterPro	http://www.ebi.ac.uk/interpro
Sequence retrieval System	http://www.expasy.org/srs
Entrez	http://www.ncbi.nlm.nih.gov/Entrez

Tabla 3.1: Lista de URL de distintas bases de datos bioinformáticas

■ **Secuencias de ADN:**

- Separar las regiones que codifican genes de las que no los codifican.
- Identificar los *intrones* (partes sin información) y *exones* (partes con información) en las secuencias de ADN de las células *eucarióticas*³.
- Predecir la formación de proteínas (estructura primaria).

■ **Secuencias de proteínas:**

- Comparar secuencias de proteínas para la búsqueda de similitudes funcionales.
- Realización de múltiples alineamientos de proteínas.
- Buscar e identificar los diferentes *dominios* en proteínas.
- Predecir la forma del pliegue de la estructura secundaria.

■ **Estructuras moleculares:**

- Alinear estructuras en tres dimensiones.
- Medir la geometría de las macromoléculas.
- Predecir y simular las interacciones entre moléculas.

■ **Genomas completos:**

- Caracterizar las repeticiones que se producen en las secuencias.
- Asignar una estructura a los genes.
- Secuenciar completamente el genoma de un organismo.
- Realizar un censo de genes a nivel genómico.

De entre todos los problemas, citados o no anteriormente, dentro del mundo de la bioinformática, este texto va a centrarse en aquellos relacionados con el estudio de secuencias de proteínas. Esto atiende a dos motivos principales:

³Aquellas que poseen núcleo celular. Las que no tienen un núcleo diferenciado se llaman *procarióticas*. En las primeras su ADN contiene fragmentos carentes de información entre los que codifican proteínas. Éstos son eliminados por el ARN mensajero al salir del núcleo celular para sintetizar la proteína. En las células procarióticas estos fragmentos *inútiles* no existen.

1. El objetivo es buscar una aplicación dentro del mundo de la bioinformática en la que pueda aplicarse el kernel de Fisher, objeto principal de estudio del presente proyecto.
2. En segundo lugar se debe a la gran importancia que tiene este tipo de datos hoy en día. Aunque la estructura secundaria (información sobre la forma de la macromolécula) contiene más información que la primaria (información sobre la secuencia de aminoácidos). Hoy en día la mayor facilidad de obtención, de uso y la gran cantidad de datos disponibles sobre secuencias de proteínas hacen que resulte más beneficioso trabajar con ellas.

3.3. Secuencias de proteínas

Antes de comenzar a trabajar con las secuencias proteicas es conveniente intentar acercarse a la forma que tienen y las funciones que realizan. El genoma de un ser vivo es la fuente de información que codifica todos los aspectos de dicho organismo. Está compuesto por cadenas de ácidos nucleicos, agrupados en cromosomas, que se encuentran dentro de todas sus células. Cada individuo posee su propio genoma, pero éste guarda gran similitud con los individuos de su misma especie y con los de especies similares⁴. Los cromosomas contienen información en forma de genes. Un gen es una secuencia específica de bases nucleicas que sirven para sintetizar una proteína. Cada grupo de tres bases de ADN (adenina, citosina, guanina y timina) forman un codón, que es sustituido por un aminoácido (tabla 3.2). Los codones también pueden codificar la posición de inicio o de fin de la proteína. Esta sucesión de aminoácidos se conoce como estructura primaria de la proteína.

Las proteínas son polímeros lineales de moléculas de α -aminoácidos. La unión entre ellos se realiza al formarse un enlace peptídico. Debemos distinguir entre oligopéptidos (menos de 12 aminoácidos), polipéptidos (entre 12 y 60 aminoácidos) y proteínas (más de 60). Las proteínas son las macromoléculas que mayor número de funciones realizan entre las que forman parte de los seres vivos. Están constituidas por átomos de carbono, hidrógeno, oxígeno y nitrógeno y por cantidades menores de azufre y fósforo. Con menor frecuencia intervienen yodo, calcio, hierro,

⁴El genoma de un ser humano es igual en un 99.8 % al de cualquier otro ser humano y tan solo se diferencia del de un chimpancé en algo más de un 1 %.

AAA: K (Lys)	GAA: E (Glu)	TAA: STOP	CAA: Q (Gln)
AAG: K (Lys)	GAG: E (Glu)	TAG: STOP	CAG: Q (Gln)
AAT: N (Asn)	GAT: D (Asp)	TAT: Y (Tyr)	CAT: H (His)
AAC: N (Asn)	GAC: D (Asp)	TAC: Y (Tyr)	CAC: H (His)
AGA: R (Arg)	GGA: G (Gly)	TGA: STOP	CGA: R (Arg)
AGG: R (Arg)	GGG: G (Gly)	TGG: W (Trp)	CGG: R (Arg)
AGT: S (Ser)	GGT: G (Gly)	TGT: C (Cys)	CGT: R (Arg)
AGC: S (Ser)	GGC: G (Gly)	TGC: C (Cys)	CGC: R (Arg)
ATA: I (Ile)	GTA: V (Val)	TTA: L (Leu)	CTA: L (Leu)
ATG: M (Met)	GTG: V (Val)	TTG: L (Leu)	CTG: L (Leu)
START			
ATT: I (Ile)	GTT: V (Val)	TTT: F (Phe)	CTT: L (Leu)
ATC: I (Ile)	GTC: V (Val)	TTC: F (Phe)	CTC: L (Leu)
ACA: T (Thr)	GCA: A (Ala)	TCA: S (Ser)	CCA: P (Pro)
ACG: T (Thr)	GCG: A (Ala)	TCG: S (Ser)	CCG: P (Pro)
ACT: T (Thr)	GCT: A (Ala)	TCT: S (Ser)	CCT: P (Pro)
ACC: T (Thr)	GCC: A (Ala)	TCC: S (Ser)	CCC: P (Pro)

Tabla 3.2: Correspondencia entre los diferentes codones y los aminoácidos

magnesio y otros elementos. Constituyen alrededor del 50% del peso de un ser humano, una vez eliminada la parte correspondiente al agua.

En la figura 3.2 puede verse la estructura de un aminoácido, la pieza básica con el que se construirán las proteínas. Existen 20 aminoácidos distintos (tabla 3.3) que pueden formar parte de las proteínas. Se conocen también alrededor de otros 200 aminoácidos más, que se encuentran en diferentes tejidos o células, pero que no forman parte de ninguna proteína.

El enlace peptídico es un enlace de tipo amida secundaria y se establece entre el grupo α -carboxilo de un aminoácido y el α -amino de otro, liberándose una molécula de agua y formándose un dipéptido (figura 3.3). Tres aminoácidos pueden unirse mediante dos enlaces peptídicos y formar un tripéptido. Así se van uniendo muchos aminoácidos, formando polipéptidos y proteínas.

Las características del enlace peptídico determinan la estructura de

3. INTRODUCCIÓN A LA BIOLOGÍA COMPUTACIONAL

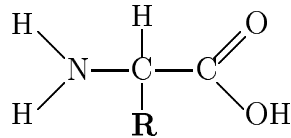


Figura 3.2: Estructura de un Aminoácido. Cada aminoácido tiene este base con radical (marcado como **R** en la figura) diferente. Existen más de 200 aminoácidos diferentes, y por lo tanto 200 radicales distintos, aunque en seres vivos aparecen 20.

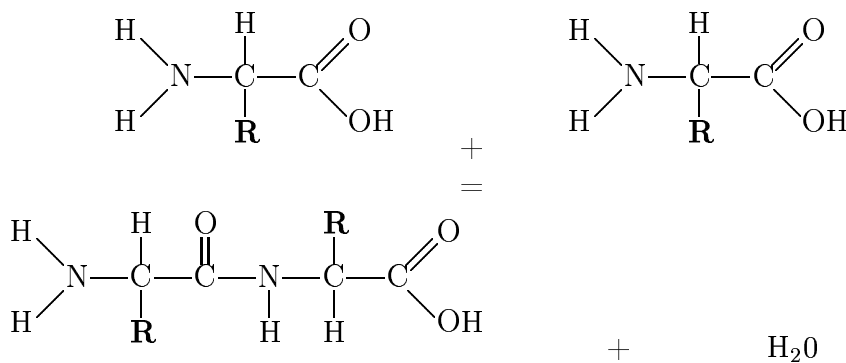


Figura 3.3: Enlace peptídico. La unión se produce entre el grupo α -carboxilo de una aminoácido y el grupo α -amino de otro. Como resultado se produce un dipéptido y una molécula de agua.

las proteínas que se pueden formar. Los átomos de la unión **C-N** se sitúan en el mismo plano debido a la estabilización por resonancia que se produce. Esto confiere al enlace un carácter parcial de doble enlace cuya rigidez no permite movimientos de rotación entre estos dos átomos. El resto de las uniones que forman los átomos de carbono, a las que pertenecen las cadenas laterales o radicales (**R**), sí que pueden girar, dando cierto grado de libertad a los pliegues que puede tomar la macromolécula. En teoría, gracias a la libre rotación, sería posible encontrar un alto número de formas o conformaciones para cualquier proteína. En la práctica esto no sucede. La mayoría se pliega adoptando una única configuración debido a las interacciones débiles entre las cadenas laterales de los aminoácidos que la forman o entre éstas y el medio circundante. Como se ha comentado anteriormente, la velocidad de descubrimiento de nuevos pliegues ha disminuido en los últimos años, aunque el número de secuencias diferentes sigue creciendo (figura 3.1).

Alamina	Ala	A
Valina	Val	V
Leucina	Leu	L
Isoleucina	Ile	I
Prolina	Pro	P
Metionina	Met	M
Fenilalanina	Phe	F
Triptófano	Trp	W
Glicocola	Gly	G
Serina	Ser	S
Treonina	Thr	T
Cisteina	Cys	C
Tirosina	Tyr	Y
Asparagina	Asn	N
Glutamina	Gln	Q
Ácido aspártico	Asp	D
Ácido glutámico	Glu	E
Lisina	Lys	K
Arginina	Arg	R
Histidina	His	H

Tabla 3.3: Lista de aminoácidos que pueden formar parte de las proteínas de los seres vivos.

Existen dos tipos de proteínas: las no activas o fibrosas, que realizan funciones estructurales y las activas o globulares, que participan en la dinámica celular, en la que desempeñan funciones de transporte, catalíticas, de defensa, etc. Estas últimas son capaces de reconocer otras sustancias, denominadas *ligandos*, a las que pueden unirse.

La estructura primaria viene determinada por los aminoácidos que componen la proteína y el orden en que se encuentran en la cadena polipeptídica. Cualquier variación de esta secuencia puede dar lugar a proteínas diferentes. Un ejemplo de esta secuencia puede ser, por ejemplo, el siguiente:

```
SLFEQLGGQAAVQAVTAQFYANIQADATVATFFNGIDMPNQTNKTA AFLC
AALGGPNAWTGRNLKEVHANMGVSNAQFTTVIGHLRSALTGAGVAAALVE
QTVAVAETVRGDVVTV
```

No tenemos más que una secuencia de letras, cada una de las cuales

3. INTRODUCCIÓN A LA BIOLOGÍA COMPUTACIONAL

identifica un aminoácido diferente (tabla 3.3).

Según la proteína se va sintetizando, las cadenas laterales de los aminoácidos hidrofóbicos tienden a agruparse en el interior de la molécula, evitando así el medio acuoso. Al mismo tiempo, la mayoría de restos polares se sitúan en la parte exterior de la molécula, donde forman puentes de hidrógeno con las moléculas de agua. Como los enlaces peptídicos también son polares, reaccionan entre sí y con otros restos polares en el interior de la molécula. De ahí el importantísimo papel que desempeñan los enlaces por puentes de hidrógeno en el mantenimiento de la estructura de las proteínas y en las interacciones que existen en la superficie de éstas. La localización espacial y las características químicas de los radicales de la superficie originan una especificidad de las proteínas con respecto a su capacidad de fijarse a otras superficies moleculares o de adherir otras moléculas a la suya. La anterior secuencia mostrada como ejemplo, una vez plegada, tendrá una estructura secundaria como la que se muestra en la figura 3.4, determinada por estas interacciones débiles.

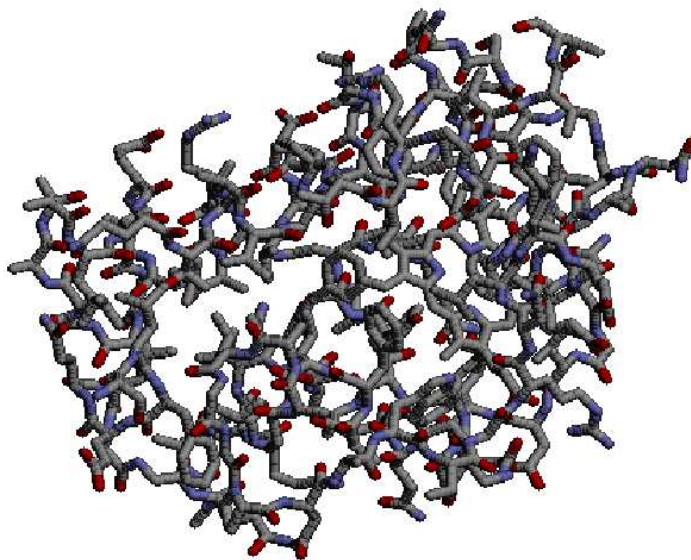


Figura 3.4: Estructura secundaria de una proteína.

Al comparar estructuras diferentes se observan plegamientos que se repiten a pesar de que la conformación final puede ser diferente en ca-

da caso. Los tipos de plegamientos, que originan los enlaces por puentes de hidrógeno formados entre los átomos de los enlaces peptídicos, son la α -hélice y la β -lamina u hoja plegada, que constituyen la llamada estructura secundaria. Los plegamientos se establecen, por tanto, en base a la naturaleza de los enlaces que se forman entre componentes de la cadena peptídica. Estos primeros niveles de plegamiento constituyen usualmente la base del siguiente nivel estructural, denominado dominio o rango. Cierta número de dominios suelen estar unidos por medio de fragmentos de cadena polipeptídica sin ordenación aparente. Ésto forma una estructura, que se mantiene por las interacciones de las cadenas laterales o residuos, e incluyen enlaces por puentes de hidrógeno y disulfuro, atracciones iónicas, interacciones hidrofóbicas y fuerzas de Van der Waals.

3.4. Clasificación de secuencias

La evolución molecular comenzó a ser estudiada en los años 60 con tan sólo unas pocas secuencias de proteínas, de varios organismos diferentes. Bajo la suposición de que organismos estrechamente relacionados poseerían secuencias similares, se construyeron una serie de familias para esas secuencias. Esta relación se basa en que la evolución molecular debería ser común para estos organismos. Si están lo suficientemente cerca en los pasos evolutivos, los cambios en las secuencias serán menores, de tal forma que podrán identificarse las similitudes ([Wat95]).

Esta teoría evolutiva puede resumirse considerando que dos secuencias similares deberían conducir a proteínas con comportamientos equivalentes. En este punto de partida tan simple, se esconde el inmenso problema de definir qué secuencias son *similares* entre sí y según que criterio lo son. La forma más sencilla de definir esta similitud es consiste en tener en cuenta cuales pueden ser los cambios que se producen en la secuencia durante la evolución molecular. Cuantos más cambios se produzcan más *diferentes* serán las secuencias. Desde aquí parten los métodos de similitud que se comentarán a continuación. Éstos fueron los primeros en emplearse y tras ellos hay una amplia base matemática.

Otra posible aproximación a la comparación de secuencias es la probabilística. En ella se construye un modelo para cada familia o clase de proteínas. Posteriormente se evalúa la probabilidad de que una secuencia determinada pertenezca a alguno de estos modelos. Esta aproximación

presenta ventajas sobre la anterior: posee la capacidad de asignar puntuaciones locales a los alineamientos y puede introducir conocimiento previo que se tenga acerca de los datos en el modelo.

Los métodos comentados a continuación pueden emplearse tanto en secuencias de ADN como de proteínas sin muchos cambios en cuanto a la algoritmia. Las diferencias radicarán en la naturaleza de estos dos tipos de secuencias. Serán necesarias consideraciones adicionales en cada caso, desde cambiar el valor de algunas de las constantes de los algoritmos hasta tener que modificar los modelos para reflejar las particularidades de cada tipo de secuencias. A pesar de las similitudes, en lo sucesivo se hablará siempre de secuencias de proteínas, por ser el caso elegido para los experimentos.

3.4.1. Métodos de similitud

Los eventos más sencillos que pueden ocurrir durante la evolución molecular son la sustitución de una base por otra, la inserción de alguna nueva o el borrado de una posición. Estos cambios pueden considerarse como una serie de errores de transcripción durante la replicación del material genético. La forma más directa de tenerlos en cuenta a la hora de comparar secuencias, consiste en contar, posición a posición, cuales son los cambios que se han producido al pasar de una secuencia a la otra. La existencia o no de estos cambios, debidamente ponderada, puede emplearse como medida de similitud entre las dos secuencias. Un heurístico que hace tal medida puede ser el siguiente:

$$S = \sum (\# \text{identidades} - \mu \# \text{sustituciones} - \delta \# \text{insercciones-borrados}) \quad (3.1)$$

Las constantes μ y δ son las penalizaciones tanto de substituciones como de borrados e inserciones. Una vez definido el valor de estas penalizaciones, se buscará cuál es el mejor alineamiento entre secuencias. Éste será el que produzca mayor similitud entre ellas. Existen multitud de formas de calcular el mejor alineamiento de manera eficiente ([NW70], [Wat95]). Los algoritmos de alineamiento más destacados en el mundo de la bioinformática son los de *Needleman-Wunsch* y de *Smith-Waterman*. En aproximaciones heurísticas de estos dos algoritmos se basan las herramientas BLAST (Basic Local Aligment Search Tool) ([AGM⁺90],[AMS⁺97]) y FastA (Fast Alignment) ([PL88]).

La ventaja que proporciona esta distancia es que permite construir un espacio métrico sobre las secuencias. Con una medida de distancia en este espacio puede verse que dadas dos secuencias a y b :

1. $D(a, b) = 0$ si y solo si a es idéntico a b .
2. $D(a, b) = D(b, a)$. La medida de distancia es simétrica.
3. $D(a, b) \leq D(a, c) + D(c, b)$ para cualquier otra secuencia c (desigualdad triangular).

Diversos algoritmos, como el de Needleman-Wunsch, construyen una matriz de distancias basadas en esta métrica que se utiliza para calcular el mejor alineamiento.

Para clasificación, se puede utilizar esta métrica como base para un algoritmo de *k vecinos cercanos* (*k-NN*), en el que se asigne una nueva secuencia al grupo de las k secuencias más cercanas, según la métrica, a ella.

3.4.2. Modelos probabilísticos

Los anteriores métodos proporcionan una medida directa de la similitud entre secuencias comparando las posiciones de las mismas. Detrás de todos estos sistemas hay una gran teoría estadística que los avala pero su principal problema es que no tienen en cuenta el hecho de que la información no tiene por qué estar igualmente distribuida a lo largo de la secuencia.

Un acercamiento estadístico a este análisis se basa en la utilización de un *modelo de probabilidad generativo*. El caso más común es emplear un modelo oculto de Markov (HMM) ([DEKM98], [KBH98], [KSB⁺97] y [KBM⁺93]). Estos modelos pueden construirse, por ejemplo, para una familia o una superfamilia de proteínas.

El modelo por sí solo no es capaz de discriminar si una proteína pertenece a una clase o no. El modelo H_1 simplemente nos dará la probabilidad de que una muestra pertenezca o no a la clase sobre la que está construido, $P(X|H_1)$. Para realizar la labor de discriminación entre las muestras que pertenecen a esta clase y las que no, no es suficiente con este valor. Será necesario construir otro modelo que indique la probabilidad de que una secuencia no pertenezca a la familia. Éste se conoce como modelo

nulo, H_0 . Una vez definidos H_1 y H_0 puede construirse una función de discriminación por máxima verosimilitud como la siguiente:

$$\mathcal{L}(X) = \log \frac{P(X|H_1)P(H_1)}{P(X|H_0)P(H_0)} = \log \frac{P(X|H_1)}{P(X|H_0)} + \log \frac{P(H_1)}{P(H_0)} \quad (3.2)$$

El signo de este coeficiente indica si una secuencia pertenece o no al modelo H_1 .

3.5. Modelos ocultos de Markov en bioinformática

Los modelos ocultos de Markov pueden aplicarse a los problemas de modelado estocástico de secuencias, la búsqueda en bases de datos y alineamientos múltiples de familias y dominios de proteínas ([Edd98b]). Su uso es interesante ya que son capaces de establecer un modelo probabilístico para una clase de datos y permiten trabajar con secuencias de cualquier tamaño sin problemas.

La bondad de los métodos de similitud, como los comentados anteriormente, depende en gran medida de una serie de parámetros que el usuario debe ajustar. Aún escogiendo correctamente todas las penalizaciones, existe una limitación intrínseca a ellos. Las penalizaciones se aplican por igual a lo largo de toda la secuencia. Cada residuo, o serie de residuos, en una secuencia funcional, puede estar sometido a diferentes presiones selectivas que los demás.

Los alineamientos múltiples de una familia de proteínas, muestran la existencia de regiones muy bien conservadas a lo largo de todos los miembros de la familia, mientras que otras parecen admitir grandes cambios. Intuitivamente resulta deseable emplear información específica de la posición cuando se realicen búsquedas de secuencias homólogas. Los métodos de perfil (*profile methods*) ([Tay86], [GME87]) se emplean para construir modelos que tengan en cuenta la información puntual de la secuencia a la hora de evaluarla.

Los modelos ocultos de Markov (HMM) proporcionan una teoría coherente para los métodos de perfil. Los HMM son un tipo de modelos probabilísticos de aplicación general para series temporales o para secuencias lineales. Fueron introducidos en el mundo de la bioinformática a finales

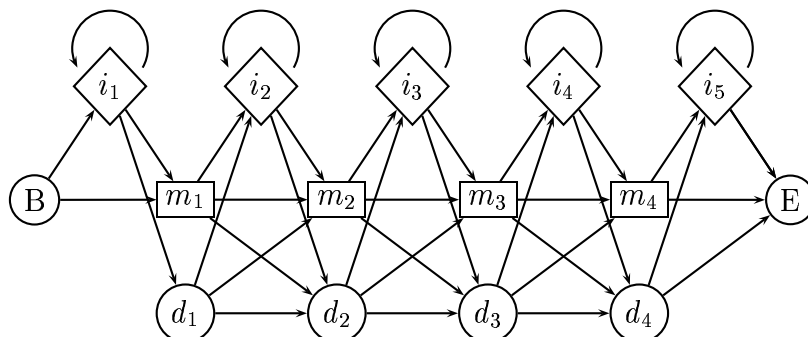


Figura 3.5: Arquitectura de un modelo oculto de Markov de perfil.

de los 80 ([Chu89]) y los modelos ocultos de Markov de perfil a mediados de los 90 ([KBM⁺93]).

La arquitectura del modelo se muestra en la figura 3.5. Se trata de una sucesión de estados de emparejamiento (m_i) que suelen corresponderse a una posición de la cadena normalmente conservada⁵. Asociados a este estado existen uno de borrado (d_i) y otro de inserción (i_i). El estado de borrado permite saltar la posición de emparejamiento a la que está asociado, lo que hace que sea posible que no todas las posiciones de emparejamiento aparezcan en la cadena. El estado no emitirá ningún residuo. Los de inserción, permiten introducir residuos entre dos posiciones de emparejamiento consecutivas, indicando la posibilidad de que aparezcan residuos entre dos estados de emparejamiento o que incluso algún estado de emparejamiento se sustituya por otra serie de residuos. Los estados de comienzo y de final (B y E en la figura) tampoco tienen asociada emisión alguna.

La razón para elegir esta arquitectura en particular y no otra, es porque es capaz de capturar la estructura de la proteína. Esto se debe principalmente a que:

- En primer lugar, se muestra una secuencia de posiciones, en cada una de las cuales habrá una probabilidad de emisión diferente.
- En segundo lugar, existe la opción de quitar o insertar un aminoácido entre posiciones consecutivas.
- Por último, permite incluso que los estados de borrado e inserción

⁵Una posición normalmente conservada puede encontrarse en las columnas de un alineamiento múltiple de secuencias.

sean más probables que el de emparejamiento.

Aun así, no se asegura en ningún caso que no exista otra arquitectura que funcione mejor para un problema en particular. La ventaja de los modelos ocultos de Markov de perfil es precisamente su generalidad, lo que les permite funcionar correctamente en un amplio conjunto de problemas.

La utilidad del modelo pasa por su capacidad de aplicación a un problema real. En este contexto debe extraerse la información mediante un proceso de entrenamiento. Los algoritmos de entrenamiento habituales como el *Baum Welch* pueden emplearse sin problemas con esta arquitectura. Para evaluar una secuencia con el modelo, podrán emplearse tanto un algoritmo *Viterbi* como un *forward-backward*.

Aunque una vez definida la arquitectura del modelo, el algoritmo de entrenamiento puede aplicarse sin problemas, existe una dificultad a la hora de averiguar cuál es el número de estados adecuado. Hay dos métodos para afrontar la búsqueda:

- En primer lugar es posible emplear datos previamente alineados con algún algoritmo de alineamiento múltiple. Utilizando esta información se asignará un estado de emparejamiento a cada posición del alineamiento. Las probabilidades del modelo se pueden asignar sin más que evaluar la frecuencia con la que aparece cada residuo en los estados. A pesar de que con este método puede realizarse el entrenamiento, el sistema adolece de una gran dependencia del correcto alineamiento de los datos. Si éste no es bueno el modelo generado tampoco lo será.
- Una alternativa para buscar cuál es la arquitectura adecuada consiste en entrenar un modelo con un número de estados cualquiera y luego ir modificándolo hasta encontrar el correcto. Cuando la probabilidad de pasar por un estado de borrado en vez de por el de emparejamiento sea superior a $1/2$ se elimina este último. Igualmente cuando la probabilidad de pasar por un estado de inserción sea superior a $1/2$ se coloca un estado de emparejamiento entre los dos unidos por el de inserción ([KBM⁺93]).

3.5.1. Enfoque discriminativo

Los parámetros de un modelo generativo se estiman de tal manera que proporcionan altas verosimilitudes a aquellas muestras que pertenecen al

grupo de entrenamiento, miembros de la clase que está siendo modelada. En contraste, los parámetros de un modelo discriminativo se eligen teniendo en cuenta tanto las muestras positivas como las negativas. El objetivo es obtener un sistema tal que la puntuación derivada de él pueda ser usada para diferenciar entre los miembros de una familia y aquellos que no lo son.

A pesar de que el cociente de verosimilitudes (expresión 3.2) es óptimo cuando los modelos H_1 y H_0 son perfectamente conocidos, puede funcionar pésimamente cuando estos modelos no son lo suficientemente exactos. Esto puede ocurrir, sencillamente, cuando el tamaño del conjunto de entrenamiento sea limitado. Los métodos discriminativos pueden emplearse para optimizar la función de decisión utilizando tanto las muestras positivas como las negativas. En general su comportamiento es mejor que el de la anterior aproximación.

Entre estos métodos discriminativos destaca el kernel de Fisher ([JH98], [JDH98] y [JDH99]) en su utilización con máquinas de vectores soporte (capítulo 2). Para la aplicación del kernel de Fisher al problema de la clasificación de proteínas no hay más que emplear las expresiones obtenidas en la sección 2.5, como se verá en el siguiente capítulo.

3. INTRODUCCIÓN A LA BIOLOGÍA COMPUTACIONAL

CAPÍTULO 4

Experimentos

La búsqueda de homologías remotas en secuencias de proteínas es un problema que se ha resuelto, habitualmente, empleando métodos de similitud, como los utilizados en el programa *Blast*. El kernel de Fisher también puede utilizarse para resolver este tipo de tareas. En este capítulo se muestran los resultados con estos dos enfoques, el tradicional y el nuevo, para un problema de este tipo.

4.1. Definición del experimento

Para validar el funcionamiento del kernel de Fisher se ha realizado una implementación del mismo que se ha aplicado a un problema de *búsqueda de homologías remotas entre proteínas*. Dos proteínas son homólogas cuando presentan tanto una secuencia como una funcionalidad similares. Cuando se encuentra una nueva proteína, con una secuencia desconocida hasta entonces, se pueden buscar secuencias homólogas a ella con el fin de extraer información acerca de su funcionalidad. Esta búsqueda se realiza en base a la similitud de secuencias, como se comentó en el capítulo 3, aunque también existen métodos que emplean modelos ocultos de Markov.

Para los experimentos que aquí se presentan, se va a abordar un problema artificial con datos reales, como el empleado en [JDH98]. No se ha utilizado un problema real, sencillamente porque implicaría la necesidad, principalmente, de contar con un equipo especializado en biología que

4. EXPERIMENTOS

interprete y avale los resultados obtenidos. Con el fin de evitar esta carencia, se empleará una base de datos, ampliamente utilizada y conocida, que proporciona, de antemano, una separación de las secuencias, atendiendo a criterios estructurales y funcionales. De esta forma se obtiene una manera sencilla de evaluar el correcto funcionamiento del sistema, al disponerse de una serie de datos previamente etiquetados.

Los datos usados se van a extraer de las base de datos *SCOP* (*Structural Classification of Proteins*). El SCOP proporciona una descripción detallada de las relaciones, estructurales y evolutivas, entre proteínas de estructura conocida. También dispone de enlaces con información de la disposición tridimensional de cada una de las proteínas de la base de datos, la secuencia de aminoácidos que la forman y referencias bibliográficas relevantes en cada caso. La base de datos es de acceso público y se encuentra disponible en la URL <http://scop.mrc-lmb.cam.ac.uk/scop>.

Todas las proteínas del SCOP están divididas, atendiendo a sus relaciones evolutivas y su estructura tridimensional, en una serie de *familias* ([MBHC95]). Estas familias se agrupan empleando dos criterios. En primer lugar se consideran todas las proteínas que tienen un 35 % de identidades en sus residuos. Posteriormente también se consideran las secuencias, que aún teniendo menos de este 35 % de identidades, presentan funciones y estructuras muy similares, como, por ejemplo, la globina, con sólo un 15 % de identidades entre secuencias. Estos dos criterios están encaminados a buscar proteínas con un origen evolutivo común. Este proceso no se realiza de forma automática sino que la última palabra sobre dónde se colocan las diferentes secuencias siempre la tiene un operario humano.

Las familias, a su vez, se agrupan en un conjunto superior denominado *superfamilia*. Las proteínas de estas superfamilias tienen pocas identidades en sus secuencias pero sus estructuras, y en muchos casos sus funciones, sugieren que es probable que compartan un origen evolutivo común.

Las superfamilias se unen en otros conjuntos denominados *pliegues*. Varias superfamilias están en un mismo pliegue cuando sus estructuras secundarias presentan disposiciones parecidas, con las mismas conexiones topológicas. Las similitudes estructurales, de las proteínas colocadas en el mismo pliegue, probablemente derivan de las características físicas y químicas de la molécula, que pueden favorecer determinadas conformaciones y topologías. Hay algunos casos, sin embargo, en los que un origen evolutivo común se ve oscurecido por una amplia divergencia en la se-

cuencia, estructura o función. En estas situaciones, el descubrimiento de nuevas estructuras, con pliegues entre los de las estructuras anteriormente conocidas, puede clarificar estos lazos evolutivos.

Una vez que se conoce esta clasificación es fácil reproducir un experimento de búsqueda de homologías remotas. Sabiendo que dos proteínas de diferentes familias que pertenecen a una misma superfamilia guardan una homología remota pueden seleccionarse estas secuencias para encontrar dichas homologías.

Para simplificar los experimentos no se utiliza la base de datos completa sino una modificación conocida como PDB90 basada en la versión 1.37 del SCOP. La parte denominada PDB90 consiste en una serie de dominios de proteínas que no tiene más de un 90 % de identidades de residuos entre dos secuencias distintas. Lo que se consigue al emplear PDB90, en vez de la base completa, es eliminar un gran número de secuencias esencialmente redundantes. Además sólo se emplearán dominios de proteínas. Los dominios son fragmentos con una función determinada dentro de las secuencias de proteínas. Una proteína se construye con la unión de diferentes dominios, separados por regiones cuya función, en principio, es difícil de precisar. Puede encontrarse una analogía a esta distinción, en la diferencia que existe, al hablar, entre las palabras y las frases que se forman con ellas. Con el fin de eliminar esta ambigüedad, que requeriría consideraciones adicionales, sólo se usarán dominios de proteínas.

Sobre la separación que realiza SCOP se forman los conjuntos de entrenamiento y de verificación o test. Se toman como secuencias de entrenamiento positivas todas aquellas que pertenecen a una misma superfamilia, eliminando completamente una de sus familias. Esta familia, la separada del resto, se usará para el conjunto de test. Las muestras negativas tanto de entrenamiento como de test se toman de familias situadas en otros pliegues. En la figura 4.1 se representa, de forma esquemática, esta separación.

Para los experimentos sólo se han seleccionado las familias que contienen por lo menos 5 secuencias, dejando al menos otras 15 secuencias en el resto de las familias de su superfamilia, siempre trabajando sobre el PDB90. Hay 32 familias de proteínas en 16 superfamilias distintas que cumplen estas restricciones. Junto con secuencias de otros pliegues se construyen los 32 conjuntos de entrenamiento y test, cuyo número de secuencias, tanto de muestras positivas como negativas, puede verse en las tablas 4.1 y 4.2.

4. EXPERIMENTOS

Identificador Secuencia	Muestras Totales	Muestras Positivas	Muestras Negativas
1.1.1.2	2257	1126	1131
1.25.1.1	1245	68	1177
1.25.1.2	1528	351	1177
1.25.1.3	1516	339	1177
1.34.1.4	3648	2497	1151
1.34.1.5	3922	2771	1151
2.1.1.1	6021	4850	1171
2.1.1.2	6217	5046	1171
2.1.1.3	6511	5340	1171
2.1.1.4	1697	526	1171
2.1.1.5	6225	5054	1171
2.19.1.1	1799	626	1173
2.31.1.1	1902	729	1173
2.31.1.2	1400	273	1127
2.34.1.1	1237	110	1127
2.41.1.1	1681	504	1177
2.5.1.1	2337	1160	1177
2.5.1.3	1398	221	1177
2.8.1.2	1436	277	1159
2.8.1.4	1362	185	1177
3.1.1.1	1555	465	1090
3.1.1.3	1420	330	1090
3.1.1.5	1479	389	1090
3.19.1.1	2022	887	1135
3.19.1.4	2024	889	1135
3.19.1.5	2014	879	1135
3.25.1.1	2389	1234	1155
3.25.1.3	1237	82	1155
3.33.1.1	1701	543	1158
3.33.1.5	1773	615	1158
3.50.1.7	2050	877	1173
3.73.1.2	1258	97	1161

Tabla 4.1: Número de secuencias en los conjuntos de entrenamiento.

Identificador Secuencia	Muestras Totales	Muestras Positivas	Muestras Negativas
1.1.1.2	1163	8	1155
1.25.1.1	1155	14	1141
1.25.1.2	1153	12	1141
1.25.1.3	1151	10	1141
1.34.1.4	1183	24	1159
1.34.1.5	1207	48	1159
2.1.1.1	1114	232	882
2.1.1.2	1191	309	882
2.1.1.3	897	15	882
2.1.1.4	898	16	882
2.1.1.5	914	32	882
2.19.1.1	1142	18	1124
2.31.1.1	1154	30	1124
2.31.1.2	1183	24	1159
2.34.1.1	1270	111	1159
2.41.1.1	1173	35	1138
2.5.1.1	1149	24	1125
2.5.1.3	1194	69	1125
2.8.1.2	1165	6	1159
2.8.1.4	1147	6	1141
3.1.1.1	1203	44	1159
3.1.1.3	1192	33	1159
3.1.1.5	1174	15	1159
3.19.1.1	1166	7	1159
3.19.1.4	1164	5	1159
3.19.1.5	1174	15	1159
3.25.1.1	1179	20	1159
3.25.1.3	1171	12	1159
3.33.1.1	1173	14	1159
3.33.1.5	1186	27	1159
3.50.1.7	1185	49	1136
3.73.1.2	1165	6	1159

Tabla 4.2: Número de secuencias en los conjuntos de test.

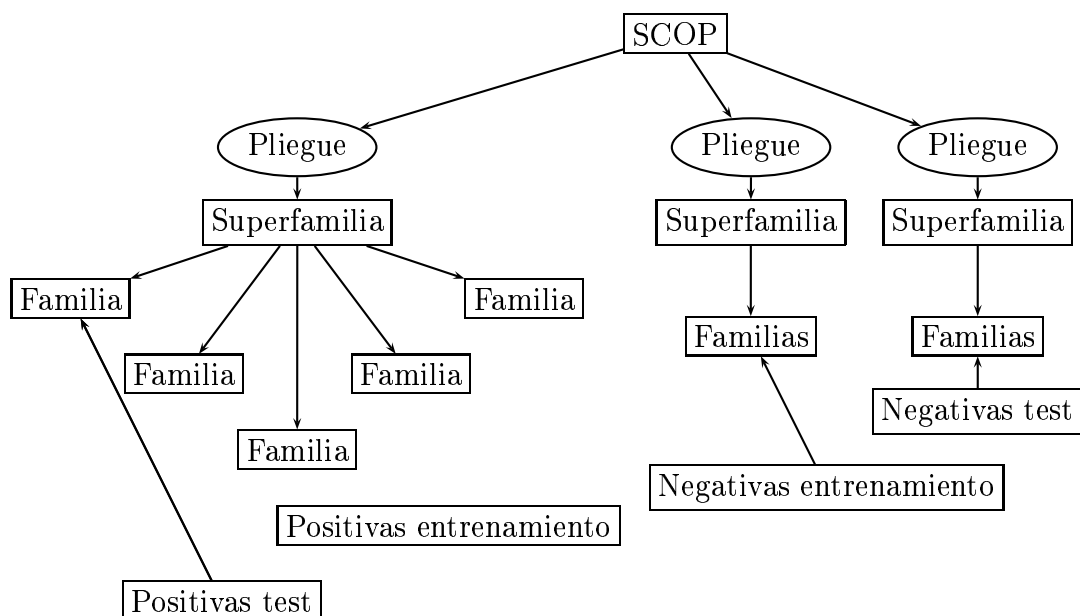


Figura 4.1: Separación de la base de datos SCOP PDB90 en los conjuntos de entrenamiento y test.

4.2. Implementación de los métodos de clasificación

Para realizar los experimentos es necesario implementar dos simulaciones distintas, el Blast y el kernel de Fisher. El primero se utilizará como nivel de referencia de los resultados obtenidos con una aplicación que habitualmente da solución a este tipo de problemas. Se usará la implementación del algoritmo `blastp` (para proteínas) del *NCBI (National Center for Biotechnology Information)*¹. Este programa emplea un algoritmo de alineamiento para buscar similitudes relevantes entre proteínas. Para emplear la misma información con los dos métodos, kernel de Fisher y Blast, se construyen bases de datos de tipo Blast con las muestras positivas y negativas del conjunto de entrenamiento. Posteriormente se realiza una búsqueda por similitud en esta base por cada muestra del conjunto de entrenamiento. Utilizando un esquema similar a la búsqueda por vecinos cercanos, se asigna cada secuencia a la clase a la que pertenezca la secuencia más parecida a ella de entre las disponibles en el conjunto de entrenamiento. En caso de no encontrar ninguna similitud relevante,

¹<http://www.ncbi.nlm.nih.gov>

se asignará la muestra a la clase negativa. Esto se hace así porque, en principio, las clase negativa puede aglutinar secuencias muy diferentes, sin similitud aparente entre sí.

En cuanto al kernel de Fisher, su implementación puede dividirse en dos partes bien diferenciadas:

- La extracción de parámetros del modelo de Markov para construir los vectores \mathbf{U}_X .
- Construir una máquina de vectores soporte cuya función de kernel utilice los vectores de puntuación de Fisher y la matriz de información de Fisher.

Para abordar la primera parte se han hecho pruebas con diferentes paquetes de modelos de Markov, centrando la búsqueda en los programas específicos para el tratamiento de secuencias de ADN y de proteínas, como puede ser el *HMMER* ([Edd98a]). Su dificultad de uso junto al hecho de que era necesario una profunda modificación del código fuente de este programa para obtener los parámetros α y β , calculados durante el entrenamiento y necesarios para calcular el vector \mathbf{U}_X , hizo que no se utilizaran estos programas en la implementación del kernel de Fisher. En su lugar se ha empleado una herramienta general para el uso y entrenamiento de modelos de Markov *HTK* [YEK⁺01]. Esta herramienta es de utilización común en el ámbito del reconocimiento de habla y tiene la ventaja de que existe buena y abundante documentación acerca de su utilización. Además permite, con modificaciones menores, obtener las matrices α y β para cada secuencia, simplificando la implementación de esta parte del kernel de Fisher. A pesar de las ventajas que HTK presenta, tiene dos deficiencias principales. En primer lugar no permite un entrenamiento discriminativo, lo que podría resultar interesante ya que el objetivo último que se pretende, al usar el modelo, es la clasificación de las secuencias. La segunda desventaja es que no puede emplearse un modelo de Markov de perfil, como los mencionados en el apartado 3.5. Estos modelos incluyen unos estados, los de borrado, que no tienen asociada emisión alguna, caso que no está considerado en HTK, excepto para el primer y último estado.

En las simulaciones se ha empleado un modelo de Markov de 11 estados como el que se muestra en la figura 4.2. Los únicos saltos permitidos son de un estado al siguiente o a él mismo. El primero y el último son estados especiales, de principio y fin respectivamente, y no tiene asociada

4. EXPERIMENTOS

emisión. Para elegir este tamaño no se ha seguido ningún procedimiento concreto, sino que se tomó, usando esta arquitectura sencilla, uno que parecía adecuado. El número de estados del modelo fijará el tamaño del vector de puntuaciones de Fisher. Por este motivo no es conveniente que sea demasiado pequeño, puesto que podríamos estar perdiendo información relevante para el posterior proceso de discriminación. Si aumentamos demasiado el número de estados existen dos problemas principales. El primero es que el modelo no se entrenará correctamente, pues el número de secuencias puede ser demasiado bajo para el número de estados. El segundo es que HTK no funciona cuando el número de estados es superior al número de elementos de una secuencia, lo que impide usar arquitecturas grandes. Al final se realizaron dos pruebas, una con 8 estados y otra con 11. Finalmente se ha seleccionado el valor mayor puesto que presentó mejores resultados sobre un conjunto de validación, formado por las familias de las tres primeras superfamilias.

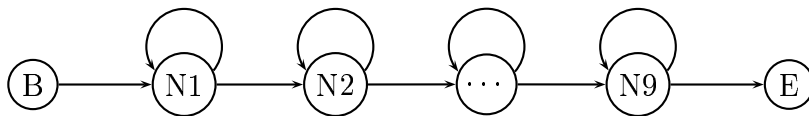


Figura 4.2: Arquitectura del modelo de Markov empleado en las simulaciones. El modelo tiene 9 estados, más el de inicio y el de fin, que no tienen asociada emisión alguna, con un total de 11.

El modelo definido anteriormente se inicializa con la herramienta de HTK *HInit*. Una vez inicializado se entrena empleando únicamente las secuencias positivas del conjunto de entrenamiento, con el programa *HRest*. Sólo se usan las muestras positivas porque HTK no permite un entrenamiento discriminativo. Tampoco se ha optado por usar todas porque en ese caso sería tan general que no describiría el proceso de generación de muestras, que es, en definitiva, la información que extrae el vector de puntuaciones de Fisher. Empleando sólo la clase positiva se intenta capturar el proceso de generación únicamente para esta clase. Al introducir una muestra de otra clase, si el modelo está bien construido, debería pre-

sentar un proceso de generación ligeramente distinto, diferencia visible a través de los vectores \mathbf{U}_X .

Una vez entrenado el modelo, se procede a extraer los parámetros α y β de cada secuencia. Esto se puede hacer realizando una iteración más en el proceso de entrenamiento que realiza *HRest*. Este paso extra se debe realizar para todas las secuencias, tanto de entrenamiento como de test, positivas y negativas. Para conseguir que *HRest* realice esta operación de la manera deseada se emplean las siguientes opciones al llamar al programa:

```
HRest -T 120 -m 1 -i 1 -M /tmp modelo.hmm secuencia.seq
```

De esta forma se utiliza el modelo *modelo.hmm*, que ha sido previamente entrenado, redirigiendo la salida a un directorio temporal, con el fin de evitar que se sobrescriba el modelo. (*-M /tmp*). Como sólo se va a emplear una secuencia es necesario ajustar el mínimo del programa a uno (*-m 1*). Además hay que fijar el número de iteraciones también a uno (*-i 1*). Con la opción *-T 120* se consigue que el programa vuelque por la salida estándar los valores de las variables α y β . Ha sido necesario, en este punto, realizar una pequeña modificación al código fuente del *HRest*, para obtener correctamente estas variables. Por defecto *HRest* sólo imprimía las 30 primeras componentes de estas matrices, por lo que ha sido necesario localizar esta limitación y eliminarla.

Una vez almacenados los valores de estas variables, para cada una de las secuencias, se procede a calcular el valor de \mathbf{U}_X . En primer lugar se obtiene la matriz $\xi(x, s)$ (expresión (2.33)) y el vector $\xi(s)$. Tomando las probabilidades de emisión (matriz B) de cada estado y pasando $\xi(x, s)$ a un vector columna, se puede calcular el vector de puntuaciones de Fisher (expresión (2.30)). Como ya se ha dicho, el primer y último estado no emiten ningún símbolo, por lo que la probabilidad de emisión no estará definida en ellos. Por lo tanto sólo se consideran los 9 estados restantes a efectos de construir el vector \mathbf{U}_X .

La matriz de información de Fisher se calcula como la correlación de los vectores \mathbf{U}_X del conjunto de entrenamiento. Los parámetros con los que se construye el kernel de Fisher son independientes entre sí, como se vio en el apartado 2.5, por lo que esta matriz de correlación será diagonal. Sólo será necesario, por tanto, calcular los valores de la diagonal, para posteriormente invertirlos y obtener F^{-1} , que es la matriz necesaria para

4. EXPERIMENTOS

calcular el kernel de Fisher:

$$K(X, Y) = \mathbf{U}_X^T F^{-1} \mathbf{U}_Y \quad (4.1)$$

Esta matriz puede dividirse, para simplificar el kernel, en dos partes iguales, con lo que éste quedará como:

$$K(X, Y) = (F^{-\frac{1}{2}} \mathbf{U}_X)^T (F^{-\frac{1}{2}} \mathbf{U}_Y) = \mathbf{U}'_X{}^T \mathbf{U}'_Y \quad (4.2)$$

Si se aplica el cambio descrito en la matriz $F^{-\frac{1}{2}}$ a todos los vectores, tanto de entrenamiento como de test, se obtiene un nuevo conjunto de datos \mathbf{U}'_X . Estos nuevos vectores pueden emplearse con un kernel tradicional, tanto lineal como gaussiano, en los que no aparece la matriz F^{-1} . Esto se puede hacer porque los vectores \mathbf{U}'_X contienen ya información de dicha matriz. La descomposición de F^{-1} en $F^{-\frac{1}{2}}$ puede hacerse fácilmente, ya que la matriz es diagonal; basta con invertir la raíz cuadrada de los valores de la diagonal. El resultado final que se obtiene al multiplicar los vectores \mathbf{U}_X por la matriz $F^{-\frac{1}{2}}$, es que cada una de sus componentes queda dividida por la desviación típica con la que se distribuyen los datos en esa variable. Esto hace que la varianza de los vectores \mathbf{U}'_X en cada componente sea 1. Éste es el ajuste que realiza la matriz de información de Fisher sobre los vectores del espacio del gradiente de la verosimilitud. Se consigue, de esta forma, que la métrica ajuste las distancias en función de la dispersión de los datos en cada una de las direcciones del espacio.

Una vez obtenidos los vectores \mathbf{U}'_X se puede emplear cualquier paquete software de máquinas de vectores soporte para realizar el aprendizaje y la clasificación. Para estas pruebas se ha empleado el programa SVM^{light} ([Joa99]), ajustando las opciones al tipo de máquina que se quiera usar.

4.3. Resultados

Una vez implementados los métodos anteriormente descritos y empleando los datos del SCOP PDB90, se procedió a realizar una serie de pruebas cuyos resultados serán presentados y discutidos a continuación.

En primer lugar se realizaron las simulaciones empleando el método Blast. Los resultados de estas pruebas se muestran en la tabla 4.3, eliminando cada una de las 32 familias seleccionadas para el experimento. En esta tabla se muestran tanto tanto la probabilidad de acierto global, como el tanto por ciento de falsos positivos y de falsos negativos sobre el número de muestras negativas y positivas respectivamente. Se

ha decidido mostrar así los resultados porque existe una gran variación entre la cantidad de muestras positivas y negativas entre los distintos subconjuntos de datos.

Una vez obtenidos los resultados con Blast, se pasa a probar el funcionamiento con el kernel de Fisher. En primer lugar se hicieron unas pruebas empleado una máquina de vectores soporte lineal, como que se muestra en la expresión (4.2). Estos resultados se muestran en la tabla 4.4.

Antes de realizar una comparativa, debe resaltarse que los resultados obtenidos no pueden compararse de forma directa. Para hacerlo sería necesario disponer de información adicional sobre el problema que indicara en qué costes o penalizaciones se incurren en los casos de aparecer un falso positivo o un falso negativo. Afortunadamente podemos comparar las prestaciones recurriendo a una gráfica del tipo de la curva de operación del receptor (*Receiver Operating Characteristic, ROC*), que pueden obtenerse a partir de la salida de la SVM. Esta curva presenta la probabilidad de falsos positivos contra la de falsos negativos. Las prestaciones de la SVM pueden moverse a lo largo de esta curva para encontrar el punto que, según las penalizaciones por cada uno de los dos tipos de errores, tiene el mínimo coste medio. Como estos valores de penalización son desconocidos, se recurre a comparar la curva ROC que proporciona la SVM, para una determinada familia, con el resultado que se obtiene con Blast. Fijando la probabilidad de un tipo de error en la curva a la que da Blast, puede observarse si la máquina se comporta mejor o peor para el otro tipo de error. Este proceso se repite para la otra clase de errores.

Puede verse un ejemplo de curva ROC en la figura 4.3. En ella se muestran las curvas, tanto sobre el conjunto de entrenamiento como el de test, para la familia 2.34.1.1. Los resultados del método Blast se pueden observar como un punto más alejado de los ejes que la curva ROC. Fijando el número de falsos positivos a un 5,34 %, el valor que proporciona Blast, la SVM obtiene un 54,05 % de falsos negativos, lo que representa un 29,73 % de mejora sobre el otro método. Fijando los falsos negativos, la mejora en falsos positivos es de un 4,91 %. Repitiendo esta operación para todas las familias se obtiene la tabla 4.5, en la que se muestra el porcentaje de errores de cada tipo para Blast menos el cometido con la máquina de vectores soporte lineal. Un signo positivo indica que el kernel de Fisher presenta mejores resultados, y el negativo lo contrario.

En la tabla se observa que el kernel de Fisher presenta un buen re-

4. EXPERIMENTOS

Familia	Probabilidad de acierto	TCFP	TCFN
1.1.1.2	91.32	8.05	100.00
1.25.1.1	98.10	0.70	100.00
1.25.1.2	94.28	4.91	83.33
1.25.1.3	94.79	4.73	60.00
1.34.1.4	86.73	13.55	0.00
1.34.1.5	86.08	14.50	0.00
2.1.1.1	68.40	39.91	0.00
2.1.1.2	69.69	40.93	0.00
2.1.1.3	59.87	40.82	0.00
2.1.1.4	93.10	6.56	25.00
2.1.1.5	60.39	40.36	18.75
2.19.1.1	93.26	5.78	66.67
2.31.1.1	90.90	7.56	66.67
2.31.1.2	92.31	6.04	87.50
2.34.1.1	87.80	5.35	83.78
2.41.1.1	80.99	17.40	71.43
2.5.1.1	73.89	26.67	0.00
2.5.1.3	95.73	4.53	0.00
2.8.1.2	79.57	20.19	66.67
2.8.1.4	95.38	4.21	83.33
3.1.1.1	71.99	27.70	36.36
3.1.1.3	76.01	23.38	45.45
3.1.1.5	73.85	25.71	60.00
3.19.1.1	70.15	30.03	0.00
3.19.1.4	69.93	30.03	40.00
3.19.1.5	70.44	29.59	26.67
3.25.1.1	78.80	20.88	40.00
3.25.1.3	94.79	4.40	83.33
3.33.1.1	88.58	11.56	0.00
3.33.1.5	87.10	12.17	44.44
3.50.1.7	76.29	24.74	0.00
3.73.1.2	93.65	6.13	50.00
Media:	82.63	17.47	41.86

Tabla 4.3: Resultados utilizando Blast. Se muestra el porcentaje de error y los tantos por ciento de falsos positivos (TCFP) y de falsos negativos (TCFN). Estas dos últimas magnitudes están normalizadas al número de muestras negativas y positivas respectivamente.

Familia	Probabilidad de acierto	TCFP	TCFN
1.1.1.2	95.79	3.55	100.00
1.25.1.1	98.18	0.61	100.00
1.25.1.2	97.83	1.84	33.33
1.25.1.3	97.39	2.10	60.00
1.34.1.4	94.93	5.18	0.00
1.34.1.5	94.37	5.87	0.00
2.1.1.1	76.48	29.48	0.86
2.1.1.2	73.47	35.83	0.00
2.1.1.3	69.34	30.84	20.00
2.1.1.4	90.53	8.28	75.00
2.1.1.5	65.97	34.35	25.00
2.19.1.1	95.80	2.67	100.00
2.31.1.1	92.46	5.78	73.33
2.31.1.2	92.98	5.87	62.50
2.34.1.1	89.76	6.30	51.35
2.41.1.1	86.96	11.69	57.14
2.5.1.1	83.20	16.62	25.00
2.5.1.3	90.20	6.93	56.52
2.8.1.2	86.09	13.55	83.33
2.8.1.4	98.17	1.31	100.00
3.1.1.1	93.93	5.61	18.18
3.1.1.3	87.50	11.04	63.64
3.1.1.5	89.10	9.75	100.00
3.19.1.1	88.85	11.22	0.00
3.19.1.4	89.26	10.53	60.00
3.19.1.5	88.93	10.96	20.00
3.25.1.1	90.50	8.80	50.00
3.25.1.3	97.10	2.07	83.33
3.33.1.1	90.03	9.40	57.14
3.33.1.5	86.93	11.04	100.00
3.50.1.7	83.88	13.12	85.71
3.73.1.2	97.94	1.55	100.00
Media:	89.18	10.43	55.04

Tabla 4.4: Resultados utilizando el kernel de Fisher con una máquina lineal. Se muestra el porcentaje de error y los tantos por ciento de falsos positivos (TCFP) y de falsos negativos (TCFN).

4. EXPERIMENTOS

Familia	Falsos positivos	Falsos negativos
1.1.1.2	8.05	0.00
1.25.1.1	0.70	0.00
1.25.1.2	4.91	66.67
1.25.1.3	4.73	40.00
1.34.1.4	12.60	0.00
1.34.1.5	10.96	0.00
2.1.1.1	8.73	0.00
2.1.1.2	5.33	0.00
2.1.1.3	1.02	0.00
2.1.1.4	-26.64	-50.00
2.1.1.5	-5.44	-6.25
2.19.1.1	-18.51	-33.33
2.31.1.1	1.33	0.00
2.31.1.2	4.75	25.00
2.34.1.1	4.92	29.73
2.41.1.1	15.47	14.29
2.5.1.1	-21.16	-12.50
2.5.1.3	-68.09	-60.87
2.8.1.2	4.06	16.67
2.8.1.4	2.03	0.00
3.1.1.1	26.75	36.36
3.1.1.3	10.44	18.18
3.1.1.5	-5.44	-20.00
3.19.1.1	27.96	0.00
3.19.1.4	10.09	0.00
3.19.1.5	19.67	20.00
3.25.1.1	10.01	0.00
3.25.1.3	3.28	16.67
3.33.1.1	-11.99	-42.86
3.33.1.5	-25.71	-55.56
3.50.1.7	-67.34	-71.43
3.73.1.2	-81.45	-50.00
Media:	-4.19	-3.73
Veces que gana Blast:	10	10
Veces que gana Fisher:	22	10
Número de Empates	0	12

Tabla 4.5: Resultados comparados con Blast de una SVM lineal.

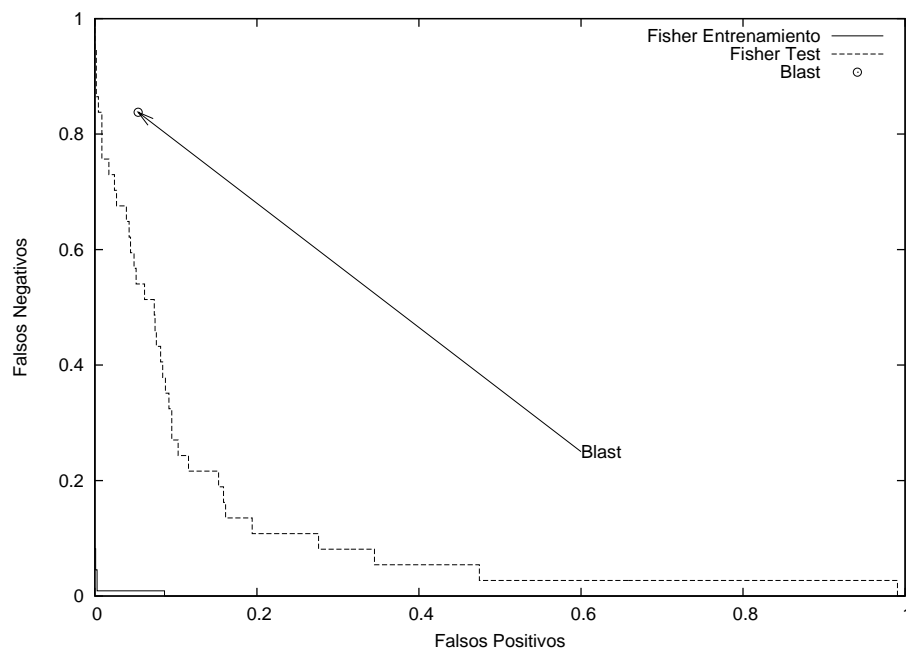


Figura 4.3: Curva de prestaciones para la familia 2.34.1.1 empleando una máquina lineal.

sultado en comparación con Blast. Aunque la media de los resultados obtenidos es negativa, Fisher es mejor, en los falsos positivos, en 22 de las 32 familias. A pesar de que en los falsos negativos presenta un empate en número de familias, puede considerarse que de forma global presenta mejores resultados que Blast. Aún así se puede intentar mejorar el sistema recurriendo a una máquina de vectores soporte con una función de kernel no lineal, como puede ser un kernel gaussiano. El valor del parámetro γ , la anchura de la gaussiana², se toma el valor típico de $\frac{1}{k}$, donde k es el número de dimensiones de los vectores de entrada. En este caso se aproxima por 0,05. Al probar con este valor se observa, sobre las familias de las tres primeras superfamilias, que los resultados nos son lo suficientemente buenos. Al comprobar la probabilidad de error sobre el conjunto de entrenamiento puede verse que esta es, en todas las familias, del 0.0%. Esto se debe a que la γ usada es demasiado alta, por lo que la fronte-

²Según la documentación, SVM^{light} toma la siguiente función de base radial:

$$\exp(-\gamma\|a - b\|^2)$$

4. EXPERIMENTOS

ra de decisión que coloca la máquina de vectores soporte, no generaliza. Para solucionarlo podemos probar con un valor 10 y 100 veces menor, para aumentar el tamaño de las gaussianas con las que se construye la frontera de decisión, buscando, de esta forma, una mejor generalización. Finalmente se ha optado por usar el valor de 0,00005 ya que presentaba mejores resultados en las seis familias de las tres primeras superfamilias, conjuntos que se usaron para la validación. Aplicando el citado valor de γ y con un valor de C igual a 100, se obtienen los resultados que se muestran en la tabla 4.6.

De nuevo pueden compararse los resultados con los obtenidos empleando el método Blast. En esta ocasión, las diferencias entre las probabilidades de falsos positivos y falsos negativos se muestran en la tabla 4.7. Puede verse que los resultados, de nuevo, son superiores a los obtenidos tanto con Blast e incluso superan a los obtenidos con un máquina lineal. En falsos positivos se consiguen las mismas prestaciones que en el caso lineal, en cuanto a familias en las que mejora, pero la media, ahora, también favorece a la SVM. En falsos negativos se ha pasado a tener 13 casos en los que funciona mejor, 9 en los que empata y 10 en los que funciona. También para falsos negativos, de nuevo, al contrario que lo que ocurría con la máquina lineal, la media también favorece a la SVM. Para comprobar los resultados obtenidos, se incluyen en el apéndice C, las curvas ROC para las 32 familias con las que se está trabajando.

Familia	Probabilidad de acierto	TCFP	TCFN
1.1.1.2	96.73	2.60	100.00
1.25.1.1	98.96	0.00	85.71
1.25.1.2	98.35	1.31	33.33
1.25.1.3	98.00	1.67	40.00
1.34.1.4	95.44	4.66	0.00
1.34.1.5	94.53	5.69	0.00
2.1.1.1	77.11	28.23	2.59
2.1.1.2	77.92	29.82	0.00
2.1.1.3	70.79	29.71	0.00
2.1.1.4	93.43	5.33	75.00
2.1.1.5	73.30	26.76	25.00
2.19.1.1	97.29	1.16	100.00
2.31.1.1	91.16	7.65	53.33
2.31.1.2	93.83	4.75	75.00
2.34.1.1	92.99	4.83	29.73
2.41.1.1	90.11	8.44	57.14
2.5.1.1	87.38	12.62	12.50
2.5.1.3	89.61	8.36	43.48
2.8.1.2	97.77	1.73	100.00
2.8.1.4	98.95	0.61	83.33
3.1.1.1	96.01	4.14	0.00
3.1.1.3	96.22	3.11	27.27
3.1.1.5	96.25	3.28	40.00
3.19.1.1	90.91	9.15	0.00
3.19.1.4	90.46	9.23	80.00
3.19.1.5	91.06	8.71	26.67
3.25.1.1	91.18	8.11	50.00
3.25.1.3	98.98	0.00	100.00
3.33.1.1	93.18	6.38	42.86
3.33.1.5	89.97	7.94	100.00
3.50.1.7	89.62	9.60	28.57
3.73.1.2	98.80	0.69	100.00
Media:	91.76	8.01	47.23

Tabla 4.6: Resultados utilizando el kernel de Fisher con una máquina gaussiana. Se muestra el porcentaje de error y los tantos por ciento de falsos positivos (TCFP) y de falsos negativos (TCFN).

4. EXPERIMENTOS

Familia	Falsos positivos	Falsos negativos
1.1.1.2	8.05	0.00
1.25.1.1	0.70	57.14
1.25.1.2	4.91	83.33
1.25.1.3	4.65	20.00
1.34.1.4	13.11	0.00
1.34.1.5	12.68	0.00
2.1.1.1	-9.18	-0.86
2.1.1.2	29.93	0.00
2.1.1.3	24.83	0.00
2.1.1.4	-24.72	-37.50
2.1.1.5	8.50	0.00
2.19.1.1	-13.97	-27.78
2.31.1.1	6.76	13.33
2.31.1.2	4.06	12.50
2.34.1.1	5.35	59.46
2.41.1.1	17.05	42.86
2.5.1.1	5.60	0.00
2.5.1.3	-59.56	-56.52
2.8.1.2	-1.04	-33.33
2.8.1.4	4.12	16.66
3.1.1.1	26.49	36.36
3.1.1.3	21.74	45.45
3.1.1.5	24.25	60.00
3.19.1.1	24.85	0.00
3.19.1.4	-5.78	-20.00
3.19.1.5	21.92	20.00
3.25.1.1	12.08	10.00
3.25.1.3	3.45	0.00
3.33.1.1	-3.71	-28.57
3.33.1.5	-23.12	-44.44
3.50.1.7	-3.52	-14.29
3.73.1.2	-42.80	-50.00
Media:	3.05	5.12
Veces que gana Blast:	10	10
Veces que gana Fisher:	22	13
Número de Empates	0	9

Tabla 4.7: Resultados comparados con Blast de una SVM no lineal.

CAPÍTULO 5

Conclusiones y trabajos futuros

A continuación se presentan las principales conclusiones obtenidas durante la realización del presente proyecto. Se analizarán críticamente los resultados presentados en el capítulo 4, intentando presentar cuales son los puntos más fuertes y más débiles del kernel de Fisher. Como último apartado se señalan las posibles líneas por las que continuar el trabajo abierto con el kernel de Fisher y sus posibles aplicaciones.

5.1. Conclusiones

Los resultados que se han obtenido durante la realización de los experimentos resultan satisfactorios en cuanto a que superan, tanto con la máquina lineal como con la no lineal, a las soluciones proporcionadas por el programa Blast. Ahora bien, a tenor de los resultados obtenidos puede pensarse que la calidad del sistema es mejorable, porque aunque supere a un método de similitud como puede ser Blast, la calidad obtenida no parece lo suficientemente alta como para competir con un modelo de perfil como *HMMER* ([Edd98a]) o *SAM-T98* ([KBH98]). Esto se debe a que el modelo de Markov utilizado puede no ser el más apropiado para el problema que se está tratando. El kernel de Fisher funciona correctamente, pero sus resultados están supeditados al correcto modelado y extracción de parámetros sobre el modelo de Markov, haciendo de éste una pieza clave a la hora de conseguir buenos resultados.

En el caso aquí planteado, los modelos de Markov de perfil ([KBM⁺93])

5. CONCLUSIONES Y TRABAJOS FUTUROS

han demostrado su correcto funcionamiento y buenas prestaciones, por lo que resulta interesante emplearlos en la construcción de un kernel de Fisher. Pueden verse resultados en esta dirección en [JDH98] y [JDH99].

En contraposición a estas consideraciones, conviene reflexionar si incluir una máquina de vectores soporte después de extraer los parámetros es útil, o si simplemente resulta suficiente trabajar con un clasificador sencillo (por ejemplo, un perceptrón). En el capítulo 4 se vio que una máquina no lineal, con más potencia, funciona mejor que una lineal, lo que indica que no se está desperdiciando la capacidad de la SVM. Aún así, existen aplicaciones en las que, por la sencillez del problema o por la buena parametrización por parte del HMM, los vectores de puntuación de Fisher son muy fácilmente separables, aún por métodos muy sencillos.

Teniendo en cuenta estas dos conclusiones, puede resumirse que aunque el funcionamiento de la máquina de vectores soporte es bueno, ésta debe basarse en un buen modelo de Markov, que proporcione una capacidad expresiva suficiente sobre el proceso de generación de la secuencia. Ahora bien, no existe una forma clara de saber cuándo un modelo es adecuado para este propósito. Lo único que en este proyecto se ha visto es que parece verosímil que una arquitectura del HMM que funcione correctamente por sí sola, vea incrementada sus prestaciones al emplear el kernel de Fisher.

Un punto que resulta interesante remarcar es qué conocimiento se ha extraído acerca del funcionamiento del kernel de Fisher. El corazón de este kernel es el vector de puntuaciones de Fisher, \mathbf{U}_X . Sobre él se construyen unas relaciones métricas ajustadas por la inversa de la matriz de información de Fisher, F , lo que confiere a las SVM el aparato matemático que necesitan para operar. Podría resultar interesante clarificar, más allá de las relaciones métricas indicadas por la matriz F^{-1} , qué representa, a un nivel conceptual, el vector \mathbf{U}_X . Recordando cómo ha sido calculado, este vector tiene información de *cuántas* veces se ha producido una emisión determina en cada uno de los estados del modelo. La diferencia de los valores en cada componente del vector de puntuaciones de Fisher, indicará qué símbolos se emiten con más frecuencia en cada estado, comparando así las dos secuencias. Para entender cómo funciona esto con el modelo de Markov con forma lineal empleado en el capítulo 4, primero debe considerarse cómo funciona dicho modelo. Al tratarse de una sucesión de estados, en el que cada uno refleja las propiedades estadísticas de una parte de la secuencia, se puede concluir que el kernel representa las diferencias de generación en cada una de estas zonas.

Indicará si hay más o menos símbolos M o T , por ejemplo, al principio de la secuencia, en el medio o al final. Esta interpretación tan simple no se podría hacer en el caso de tener un modelo más complicado, con más conexiones entre estados. Aún así, puede intuirse un comportamiento similar, buscará las diferencias en los símbolos presentes en cada una de las zonas representadas por la distribución de estados. En cuanto a la utilidad de la matriz de información de Fisher, F^{-1} , puede deducirse fácilmente que su finalidad consiste en ajustar la varianza de cada una de las dimensiones del vector \mathbf{U}_X , consiguiendo una métrica regularizada, invariante a los diferentes tamaños que de cada una de las dimensiones que forman el vector de puntuación de Fisher.

5.2. Líneas futuras de trabajo

Para continuar el trabajo expuesto en el presente proyecto se pueden tener en cuenta una serie de líneas que se han ido revelando como interesante durante la realización del mismo. En primer lugar podría realizarse un cambio en cuanto al software para los modelos ocultos de Markov. Las mejoras que pueden introducirse aquí son principalmente dos:

- Utilizar un modelo discriminativo de entrenamiento, que tenga en cuenta tanto las muestras positivas como las negativas a la hora de construir el modelo.
- Emplear un modelo de perfil como los descritos en [KBM⁺93]. Este tipo de modelos ha sido utilizado con éxito para modelar familias de proteínas, por lo que resulta interesante ver que prestaciones se alcanzan al emplear el kernel de Fisher sobre un modelos de Markov, presumiblemente, mejor usado en este proyecto.

Otra posible área de estudio consiste en desarrollar el gradiente del kernel de Fisher no sólo respecto a las probabilidades de emisión conjuntas, sino también con respecto a las probabilidades de transición. De esta forma se obtendría información acerca de cuál es el camino seguido durante la generación. Esta información puede ser interesante, si el modelo es lo suficientemente expresivo, a la hora de diferenciar entre los procesos de generación de diferentes clases de secuencias.

Por último, se puede aplicar el kernel de Fisher a otros problemas en los que intervengan secuencias de otro tipo, como puede ser voz, textos,

5. CONCLUSIONES Y TRABAJOS FUTUROS

reconocimiento de escritura, contornos de hojas de árboles, etc. Para diferentes ámbitos, fuera de la bioinformática, ya se está empezando a utilizar el kernel de Fisher. Recientes trabajos se están centrando en la aplicación del kernel de Fisher para el reconocimiento de hablantes ([SG02], [QB]) y para el análisis y categorización de textos ([SSTV], [Hof00]).

APÉNDICE A

Máquinas de Vectores Soporte para clasificación

A.1. Descripción del problema

La clasificación, en el ámbito del aprendizaje máquina, consiste en encontrar una regla de decisión que, dada una observación externa, permita asignarla a un conjunto entre varios posibles. El supuesto más sencillo es el binario. En él solo tenemos dos posibles conjuntos o clases, que denominaremos $+1$ y -1 .

La búsqueda de la regla de decisión apropiada, puede interpretarse como la estimación de una función f que asigne a cada punto del espacio de observación (de dimensión N) un punto en el espacio de las clases (-1 y $+1$ cuando sólo existan dos clases):

$$f : \mathbb{R}^N \mapsto \mathbb{Y}, \quad \mathbb{Y} = \{-1, +1\} \quad (\text{A.1})$$

La búsqueda se realiza utilizando un conjunto de datos de entrenamiento compuesto por n pares de muestras convenientemente etiquetadas, independientes e idénticamente distribuidas, generadas mediante una distribución de probabilidad desconocida $P(\mathbf{x}, y)$:

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^N, y_i \in \mathbb{Y} \quad (\text{A.2})$$

El objetivo es que la función f clasifique correctamente las observaciones que no pertenezcan al conjunto de entrenamiento, lo que se conoce

A. MÁQUINAS DE VECTORES SOPORTE PARA CLASIFICACIÓN

como generalización. Dichas observaciones se asignarán a la clase $+1$ cuando $f(\mathbf{x}) > 0$ y a la clase -1 en caso contrario. Estas muestras deben estar generadas con la misma distribución de probabilidad $P(\mathbf{x}, y)$ que la que generó las muestras de entrenamiento.

La mejor función f para llevar a cabo la clasificación será aquella con la esperanza del error de clasificación más baja, aquella con la que obtenga el mínimo coste:

$$\mathcal{C}(f) = \int l(f(\mathbf{x}), y) dP(\mathbf{x}, y) \quad (\text{A.3})$$

donde l , conocida como función de coste, expresa la penalización en la que se incurre al tomar, a partir de la observación \mathbf{x} , una decisión basada en el valor de $f(\mathbf{x})$ cuando el valor correcto es y .

Un ejemplo sencillo de función de coste es:

$$l(f(\mathbf{x}), y) = \Theta(-yf(\mathbf{x})) \quad (\text{A.4})$$

donde Θ es la función de Heaviside y cumple que $\Theta(z) = 0$ para $z < 0$ y $\Theta(z) = 1$ para el resto de los casos. Obsérvese que, según esta función de coste, una equivocación tiene un coste unidad, mientras que un acierto tiene un coste nulo.

La densidad de probabilidad $P(\mathbf{x}, y)$ es desconocida, por lo que la función de coste no puede minimizarse de forma directa empleando la expresión (A.3). Debe encontrarse una estimación de la expresión f lo más próxima posible a la óptimo, es decir, la mínimo coste. Para ello se parte de un conjunto de muestras de entrenamiento, junto con las propiedades de la familia de funciones F entre las que se busca f .

La única información que se dispone a priori es el conjunto de muestras de entrenamiento. Sobre éste podemos calcular el coste empírico para una función f :

$$\mathcal{C}_{emp}(f) = \frac{1}{n} \sum_{i=1}^n l(f(\mathbf{x}_i), y_i)$$

El coste empírico puede emplearse para tabular el coste real, sobre la distribución $P(\mathbf{x}, y)$, con una probabilidad de $1 - \eta$, $0 \leq \eta \leq 1$ ([Vap95]):

$$\mathcal{C}(f) \leq \mathcal{C}_{emp}(f) + \sqrt{\left(\frac{h(\log(2l/h) + 1) - \log(\eta/4)}{l} \right)} \quad (\text{A.5})$$

donde l es el número de observaciones y h es un entero, no negativo, conocido como la *dimensión Vapnik Chervonenkis (VC)*. Esta dimensión mide la capacidad de separación de la familia de funciones f . La teoría de Vapnik sobre la reducción del coste y la dimensión VC ([Vap95]) indica que reduciendo el coste empírico también se reduce el coste sobre la distribución $P(\mathbf{x}, y)$. La teoría de las máquinas de vectores soporte, que a continuación se presenta, parte de este punto. Intenta buscar una función f que minimice el coste empírico sobre el conjunto de entrenamiento, manteniendo a su vez la correcta generalización de la máquina.

A.2. Máquinas de vectores soporte para problemas linealmente separables

A la hora de minimizar la función de coste, el caso más sencillo que puede darse es aquel en que los conjuntos de muestras o clases son linealmente separables. Para simplificar este desarrollo se supondrá que estamos ante este caso, extendiéndolo posteriormente al caso en el que las clases no puedan ser linealmente separables. Se parte de una base de datos etiquetada, como ya se ha indicado:

$$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n) \in \mathbb{R}^N \times \mathbb{Y}, \quad \mathbb{Y} = \{-1, +1\} \quad (\text{A.6})$$

Dado que la hipótesis de partida es que las clases son linealmente separables, existen infinitos hiperplanos en \mathbb{R}^N tal que separen las muestras de una clase de las de otra. Los puntos del espacio que caen dentro de cada uno de estos hiperplanos son los que satisfacen expresiones como la siguiente:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \quad (\text{A.7})$$

donde \mathbf{w} son los vectores normal a cada hiperplano, $|b|/\|\mathbf{w}\|$ es la distancia de estos al origen y $\|\mathbf{w}\|$ la norma euclídea del vector \mathbf{w} . Dicho hiperplano separará las muestras de una clase de las de la otra:

$$\mathbf{x}_i \cdot \mathbf{w} + b > +1 \quad \text{para} \quad y_i = +1 \quad (\text{A.8})$$

$$\mathbf{x}_i \cdot \mathbf{w} + b < -1 \quad \text{para} \quad y_i = -1 \quad (\text{A.9})$$

A. MÁQUINAS DE VECTORES SOPORTE PARA CLASIFICACIÓN

Estas dos expresiones pueden combinarse en una única, más sencilla, de la siguiente forma:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad \forall i \quad (\text{A.10})$$

De entre los infinitos hiperplanos que cumplen esta condición en un problema linealmente separable, se busca aquel que tenga un margen mayor. El margen de separación será la distancia entre los dos hiperplanos que contengan a las muestras más cercanas a la frontera (ver figura A.1). Los puntos que caen sobre cada uno de los hiperplanos serán los que cumplan:

$$\mathbf{x}_i \cdot \mathbf{w} + b = 1 \quad (\text{A.11})$$

$$\mathbf{x}_j \cdot \mathbf{w} + b = -1 \quad (\text{A.12})$$

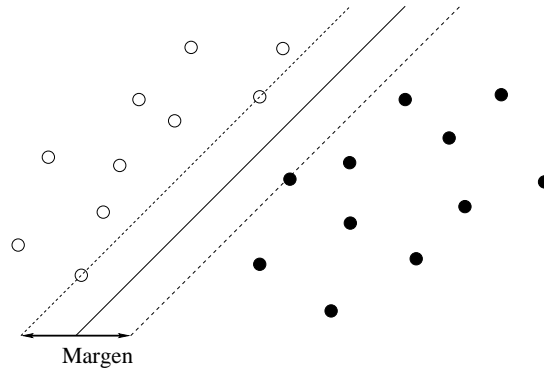


Figura A.1: Máximo margen en un problema de clasificación.

Estos hiperplanos serán paralelos al de separación, que se encontrará entre los dos. La distancia de cada uno de ellos al origen será:

$$\frac{|1 - b|}{\|\mathbf{w}\|}, \quad \text{para hiperplano de muestras positivas} \quad (\text{A.13})$$

$$\frac{|-1 - b|}{\|\mathbf{w}\|}, \quad \text{para hiperplano de muestras negativas} \quad (\text{A.14})$$

Por la tanto su distancia hasta la frontera de separación entre clases no será más que $1/\|\mathbf{w}\|$. El margen es el doble de esta distancia, $2/\|\mathbf{w}\|$.

Obsérvese que dentro de este espacio de separación no puede caer ninguna muestra, ya que se ha tomado los conjuntos de muestras \mathbf{x}_i y \mathbf{x}_j como aquellas que se encuentran más cercanas al hiperplano de separación.

De esta forma se consigue maximizar el margen (figura A.1), máximo que se alcanzará cuando $\|\mathbf{w}\|^2$ sea mínima, sujeta a las restricciones anteriormente citadas (expresión A.10). Para buscar los pesos \mathbf{w} y el hiperplano que las cumplen se emplea una formulación con *multiplicadores de Lagrange*. Uno de los motivos para emplear este método es que se pueden introducir de manera sencilla las restricciones que figuran en la expresión (A.10), simplificando los cálculos. Durante el desarrollo se encontrará que con este enfoque las muestras no aparecen más que en forma de producto interno entre vectores, lo que será de gran importancia en la posterior generalización a problemas no lineales (sección A.4) y constituye otro de los motivos para emplear este método.

Se introducen, entonces, los *multiplicadores de Lagrange* α_i , $i = 1, \dots, n$, $\alpha_i \geq 0$, uno por cada una de las restricciones del problema (expresión A.10), obteniendo:

$$L_P = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^l \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^l \alpha_i \quad (\text{A.15})$$

A continuación se procede a buscar la solución que minimice L_P con respecto a \mathbf{w} y b y, simultáneamente, cumpla que las derivadas de L_P con respecto a todos los α_i sean nulas, sujetas a $\alpha_i \geq 0$. Formulando así tenemos un problema de programación cuadrática, con las propiedades características de estos ([Bur98]). La función a minimizar es convexa y los puntos que satisfacen las restricciones forman a su vez un conjunto convexo¹. Esto indica que podemos, de forma alternativa, recurrir a una formulación dual del problema: maximizar L_P sujeto a que su gradiente con respecto a \mathbf{w} y b sea nulo y que $\alpha_i \geq 0$. Esta formulación, conocida como la formulación dual de Wolfe ([Fle87]), tiene la particularidad de que su máximo sujeto a estas últimas restricciones ocurre para el mismo conjunto de \mathbf{w} , b y α_i que el mínimo de L_P sujeto a las restricciones originales.

Haciendo que el gradiente de L_P con respecto a \mathbf{w} y b se anule se obtienen las siguientes restricciones:

¹Cualquier restricción lineal define un conjunto convexo. Un grupo de N restricciones lineales define la intersección de N conjuntos convexos, que a su vez es también convexo.

A. MÁQUINAS DE VECTORES SOPORTE PARA CLASIFICACIÓN

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \quad (\text{A.16})$$

$$\sum_i \alpha_i y_i = 0 \quad (\text{A.17})$$

Podemos sustituir estas dos expresiones en (A.15), obteniendo la siguiente formulación dual del problema:

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (\text{A.18})$$

Las dos formulaciones, L_P y L_D , son equivalentes, construyéndose a partir de las mismas restricciones iniciales. Para resolver el problema planteado se puede tanto minimizar L_P como maximizar L_D .

Hasta ahora las condiciones que se han enunciado ha sido las siguientes²:

$$\frac{\partial}{\partial \omega_\nu} L_P = \omega_\nu - \sum_i \alpha_i y_i x_{i\nu} = 0 \quad \nu = 1, \dots, d \quad (\text{A.19})$$

$$\frac{\partial}{\partial b} L_P = - \sum_i \alpha_i y_i = 0 \quad (\text{A.20})$$

$$y_i (\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad i = 1, \dots, n \quad (\text{A.21})$$

$$\alpha_i \geq 0 \quad \forall i \quad (\text{A.22})$$

$$\alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1) = 0 \quad \forall i \quad (\text{A.23})$$

Puede verse que resolver el problema de las SVM es equivalente a encontrar una solución a estas últimas expresiones. Esto proporciona diversas formas de abordar el problema, según el camino que elegido.

Los pesos \mathbf{w} están, como puede verse, determinados de manera explícita por el entrenamiento pero no así el parámetro b , que se encuentra de manera implícita. Sin embargo es sencillo obtener b usando la expresión (A.23). Eligiendo cualquier muestra i para la cual $\alpha_i \neq 0$ se despeja automáticamente el valor del parámetro b . Estas muestras serán las que

²Conocidas como condiciones *Karush-Kuhn-Tucker (KKT)*, juegan un papel central tanto en la teoría como en la práctica de la optimización sujeta a restricciones ([Fle87]).

caigan sobre los hiperplanos de separación, aquellas que cumplan que $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 = 0$. Para calcular \mathbf{w} (expresión (A.16)) es necesario y suficiente conocer estas muestras, por lo que definirán de manera unívoca la frontera de decisión. Si alguna de estas no estuviera presente, dicha frontera cambiaría. Dada su importancia, estas muestras son conocidas con el nombre de *vectores soporte*. Si falta una muestra con $\alpha_i = 0$ es sencillo ver que la frontera permanecerá inalterable.

Hasta ahora todo lo que se ha hecho es trabajar con el problema de optimización para conseguir una formulación con la que sea sencillo trabajar. Para encontrar la solución concreta en un problema real seguramente será necesario recurrir a métodos numéricos, cuya discusión no entra dentro del ámbito de esta introducción.

A.3. Máquinas de vectores soporte para casos no separables

Hasta ahora se ha supuesto que los datos pueden ser separables linealmente. De esta forma se podían definir los hiperplanos paralelos a la frontera que daban el máximo margen de separación entre clases. En un caso más general esta condición no se cumplirá, invalidando, a priori, el anterior desarrollo. Para poder abordar estos casos se introducen unas variables positivas de ajuste, $\xi_i, i = 1, \dots, n$, en las restricciones para suavizarlas, lo que las deja de la siguiente manera:

$$\mathbf{x}_i \cdot \mathbf{w} + b \geq +1 - \xi_i \quad \text{para } y_i = +1 \quad (\text{A.24})$$

$$\mathbf{x}_i \cdot \mathbf{w} + b \leq -1 + \xi_i \quad \text{para } y_i = -1 \quad (\text{A.25})$$

$$\xi_i \geq 0 \quad \forall i \quad (\text{A.26})$$

Las muestras que caigan dentro del margen de separación serán corregidas con estas variables ξ_i . De nuevo se pueden juntar estas dos restricciones en una única expresión:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0 \quad \forall i \quad (\text{A.27})$$

Con esta modificación, para que exista un error en el conjunto de entrenamiento, la ξ_i de la muestra en cuestión debe ser superior a la unidad. Estas variables estarán de esta forma relacionadas con el número de

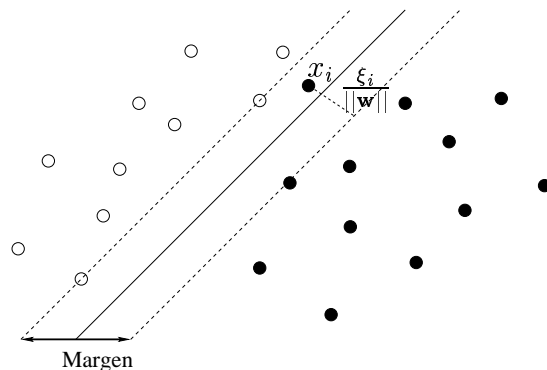


Figura A.2: Margen en caso no separable

errores en el conjunto de entrenamiento. De hecho es fácil ver que puede acotarse superiormente el número de errores que se produzcan en el conjunto de entrenamiento por el sumatorio $\sum \xi_i$. Utilizando esta particularidad se puede asignar un coste extra a los errores que se producen al caer las muestras de entrenamiento en el interior del margen de separación. En vez de minimizar simplemente $\|\mathbf{w}\|^2/2$ sujeta a las restricciones comentadas, lo que se busca es un mínimo para $\|\mathbf{w}\|^2/2 + C \sum_i \xi_i$. Esta expresión estará sujeta a las mismas restricciones que antes junto con $\xi_i \geq 0$. C indica el compromiso entre el tamaño del margen de separación y número de errores que se producen, y es un parámetro a ajustar por el usuario. Así, para un valor de C grande indica una mayor penalización de los errores a la hora de buscar los hiperplanos de separación. Con una C pequeña se obtendrá un gran margen de separación pero puede que en él caigan muchas muestras. Todo ello afectará a la generalización de la SVM siendo necesario buscar un compromiso entre el tamaño del margen y las muestras que caen en él.

Introduciendo estas nuevas restricciones, la formulación dual de Wolfe quedará de la siguiente forma:

Maximizar:

$$L_D \equiv \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (\text{A.28})$$

sujeito a:

$$0 \leq \alpha_i \leq C, \quad (\text{A.29})$$

$$\sum_i \alpha_i y_i = 0. \quad (\text{A.30})$$

En la que solo ha cambiado la expresión (A.29), limitando el valor máximo que pueden tomar las variables α_i a C (ver figura A.2). El \mathbf{w} , solución del problema, quedará ahora de esta forma:

$$\mathbf{w} = \sum_{i=1}^{N_S} \alpha_i y_i \mathbf{x}_i \quad (\text{A.31})$$

donde N_S es el número de vectores soporte. En notación primaria se obtendría:

$$L_P = \frac{1}{2} \|w\|^2 + C \sum_i \xi_i - \sum_i \alpha_i [y_i (\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i] - \sum_i \mu_i \xi_i \quad (\text{A.32})$$

Se han introducido aquí unos nuevos multiplicadores, μ_i . Estos se usan para forzar que los ξ_i sean positivos. En este caso las condiciones asociadas a L_P quedarán de la siguiente forma:

$$\frac{\partial}{\partial \omega_\nu} L_P = \omega_\nu - \sum_i \alpha_i y_i x_{i\nu} = 0 \quad \nu = 1, \dots, d \quad (\text{A.33})$$

$$\frac{\partial}{\partial b} L_P = - \sum_i \alpha_i y_i = 0 \quad (\text{A.34})$$

$$\frac{\partial}{\partial \xi_i} L_P = C - \alpha_i - \mu_i = 0 \quad (\text{A.35})$$

$$y_i (\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0 \quad i = 1, \dots, n \quad (\text{A.36})$$

$$\xi_i \geq 0 \quad \forall i \quad (\text{A.37})$$

$$\alpha_i \geq 0 \quad \forall i \quad (\text{A.38})$$

$$\mu_i \geq 0 \quad \forall i \quad (\text{A.39})$$

$$\alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i) = 0 \quad \forall i \quad (\text{A.40})$$

$$\mu_i \xi_i = 0 \quad \forall i \quad (\text{A.41})$$

De nuevo se pueden emplear las condiciones (A.40) y (A.41) para determinar el valor de b . La ecuación (A.35) combinada con la (A.41) muestran que $\xi_i = 0$ cuando $\alpha_i < C$. Para calcular b se puede tomar cualquier muestra de entrenamiento para la cual $0 < \alpha_i < C$ y usar la ecuación (A.40) teniendo en cuenta que $\xi_i = 0$.

A.4. Generalización a problemas no lineales

Aun con la salvedad de introducir las variables ξ_i , todavía no se ha realizado una extensión al caso no lineal de las máquinas de vectores soporte. Lo primero que se ha de considerar es la posible existencia de una frontera no lineal entre las clases que forman el problema. Como ya se adelantó anteriormente, en esta situación es importante que los datos aparezcan sólo como parte de producto interno, $(\mathbf{x}_i \cdot \mathbf{x}_j)$, en las expresiones anteriores.

Puede pensarse un caso tal que los datos no se empleen directamente, sino que, previamente a su utilización, sean proyectados sobre un espacio euclídeo, de dimensión cualquiera (incluso infinita).

$$\Phi : \mathbb{R}^d \mapsto \mathbb{H} \tag{A.42}$$

Las muestras una vez proyectadas, $\Phi(\mathbf{x})$, pueden usarse como un nuevo conjunto de entrenamiento. De esta forma se buscará una frontera lineal en el espacio \mathbb{H} , como se ha descrito hasta ahora. Sea cual sea el número de muestras (finito) del conjunto de entrenamiento siempre puede encontrarse un espacio de dimensión tal que permita hacer una separación lineal de las mismas. Dicha frontera definirá otra en el espacio de partida, \mathbb{R}^d , cuya forma dependerá de la función de proyección Φ empleada y de las muestras del conjunto de entrenamiento.

Como se ha hecho notar ya, los datos en la formulación de las SVM sólo aparecen como producto entre muestras, ahora $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$. En este caso sólo necesitaríamos una función, denominada *kernel*, tal que

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \tag{A.43}$$

Teniendo esta función, se puede aplicar el algoritmo de entrenamiento de las SVM sin conocer explícitamente cual es la transformación Φ o incluso el espacio \mathbb{H} . La única condición necesaria es que el kernel usado esté correctamente definido, punto que se tratará más adelante. La nueva

notación simplemente sustituirá estos productos entre muestras por la función de kernel.

Como ejemplo de una función de kernel se puede tomar la siguiente expresión correspondiente a un kernel gaussiano:

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}} \quad (\text{A.44})$$

En este caso concreto el espacio de proyección, \mathbb{H} , es infinito, imposibilitando el trabajo directo con $\Phi(\mathbf{x})$. Sin embargo, si en el anterior desarrollo se cambian todos los productos internos entre dos muestras por la función de kernel, la máquina sigue funcionando correctamente. Definirá una frontera en el espacio \mathbb{H} , que a su vez definirá otra frontera sobre el espacio de partida, \mathbb{R}^d .

Trabajando con el kernel, no será posible calcular el vector \mathbf{w} como se hacía anteriormente. En realidad dicho vector ahora no tiene sentido porque puede que el espacio \mathbb{H} o la transformación Φ sean desconocidos. Aun así se puede seguir empleando la SVM, volviendo a hacer uso de los vectores soporte. No habrá más que buscar el signo de la siguiente expresión:

$$f(x) = \sum_{i=1}^{N_S} \alpha_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) + b = \sum_{i=1}^{N_S} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \quad (\text{A.45})$$

A.5. Funciones de kernel

La función de kernel se muestra como una pieza clave en la aplicación de la teoría las máquinas de vectores soporte a los problemas no lineales. Teniendo tal importancia, es necesario detenerse en las propiedades que debe tener una función de kernel. Como se ha visto (expresión (A.43)), esta función no representa más que el cálculo de un producto interno en un espacio, posiblemente desconocido, \mathbb{H} . Dado que puede que no se conozca la función de proyección, lo que se debe resolver es cómo distinguir una función de kernel válida de una que no lo sea.

La condición de Mercer resuelve este interrogante. Esta condición indica cuando una función puede ser empleada como un kernel y cuándo. Sin embargo esta condición no aporta información sobre cómo construir la función Φ o cuál es el \mathbb{H} .

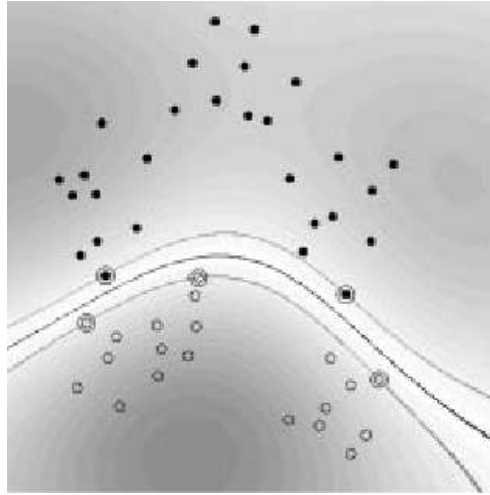


Figura A.3: Frontera no lineal de una SVM. El fondo de la imagen indica la forma de la superficie de decisión.

La condición de Mercer puede enunciarse de la siguiente forma:

Condición de Mercer:

Existe una proyección Φ y una expansión:

$$K(\mathbf{x}, \mathbf{y}) = \sum_i \Phi(\mathbf{x})_i \Phi(\mathbf{y})_i$$

si y solo si, para cada $g(\mathbf{x})$ tal que

$$\int g^2(\mathbf{x}) d\mathbf{x} < \infty$$

entonces

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0$$

Se puede encontrar una demostración de la anterior condición, por ejemplo, en [Vap95] y en [Bur98].

Al usar un kernel que no cumpla la condición de Mercer, no estamos asegurando que el problema de programación cuadrática que tenemos que resolver tenga solución. Aun así, puede darse el caso de trabajar correctamente con un conjunto de datos para los cuales la hessiana no esté indefinida, y que por tanto, la máquina funcione bien. Aunque la solución sea satisfactoria, en este caso en particular la interpretación geométrica que estamos dando a la función de kernel no tiene sentido.

A.5.1. Funciones de kernel habituales

Existen multitud de funciones de kernel, casi tantas como problemas a tratar. Las siguientes expresiones:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p \quad (\text{A.46})$$

$$K(\mathbf{x}, \mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}} \quad (\text{A.47})$$

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x} \cdot \mathbf{y} - \delta) \quad (\text{A.48})$$

se corresponden con un kernel polinómico de grado p (A.46), uno gaussiano (A.47) y uno tangencial³ (A.48). Estos kernel fueron los primeros en emplearse en los problemas de reconocimiento de patrones, por lo que son ampliamente conocidos.

³Hay que hacer hincapié en que el kernel tangencial sólo cumple la condición de Mercer para unos determinados conjuntos de parámetros κ y δ . Esto ha de tenerse en cuenta a la hora de trabajar con este tipo de kernel.

A. MÁQUINAS DE VECTORES SOPORTE PARA CLASIFICACIÓN

APÉNDICE B

Modelos Ocultos de Markov

B.1. Modelado de fuentes de información con Modelos de Markov

La representación de señales del mundo real es un problema de gran interés para multitud de aplicaciones. Aún de más provecho puede ser hacerlo en función de un modelo. Esto se debe principalmente a que éste es capaz de proporcionar los cimientos para una descripción teórica, que puede usarse para descripción y estudio de sistemas de procesamiento de señales. Otra razón muy importante es que los modelos permiten extraer información acerca de la fuente de la señal sin ser necesario disponer de la misma, e incluso permiten realizar simulaciones de fuentes, con las ventajas que ello conlleva. Por último, estos modelos han demostrado su buen funcionamiento en situaciones reales, lo que les confiere un interés muy especial.

Uno de los principales sistemas utilizados para modelar fuentes de información se conoce como *modelos ocultos de Markov* (*Hidden Markov Model, HMM*) ([Rab89]). Éstos proporcionan buenos resultados en muchas aplicaciones dado el carácter general con el que están concebidos.

Los modelos de Markov son unos sistemas caracterizados por encontrarse en cualquier tiempo en uno de sus N posibles estados. En tiempos discretos el sistema salta desde un estado a otro (que puede ser incluso el mismo) en función de un conjunto de probabilidades asociadas a cada estado. Un modelo completo debería tener en cuenta toda la secuencia

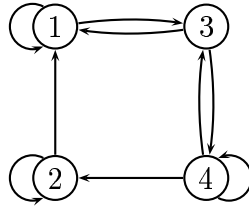


Figura B.1: Ejemplo de arquitectura de un modelo de Markov

hasta un determinado instante, pero en los de Markov sólo intervienen el estado actual y el último, es más, considera estas probabilidades de salto entre estados como constantes e independientes del tiempo. Estas probabilidades de saltos, denominadas *probabilidades de transición*, a_{ij} , indican cuál es la probabilidad pasar al estado j desde el estado i .

$$a_{ij} \geq 0; \quad \sum_i a_{ij} = 1 \quad (\text{B.1})$$

El modelo estocástico definido por las variables a_{ij} es un modelo visible de Markov. En él las salidas no son más que la secuencia de estados que se van tomando en cada instante, dado que cada uno corresponde a un evento observable.

Este modelo sería demasiado restrictivo en algunas situaciones puesto que puede darse el caso en el que sea imposible definir estados para cada evento que se observe. Podrían considerarse casos en los que las observaciones sean una función probabilística del estado en el que se encuentre y, en particular, independiente del tiempo. De esta forma se introducen los *modelos ocultos de Markov (HMM)*, en los que hay un proceso estocástico oculto (la secuencia de estados) y sobre él otro observable (la secuencia de símbolos). En el caso de que los símbolos observables sean discretos, las probabilidades de emisión se indican con la variable $b_j(k)$. Esta será la probabilidad de emitir el símbolo v_k dado que estamos en el estado s_j .

$$b_j(k) \geq 0; \quad \sum_k b_j(k) = 1 \quad (\text{B.2})$$

Si las observaciones son continuas, podría definirse su función densidad de probabilidad como una mezcla de gaussianas. En ese caso los parámetros del modelo serán la media y la varianza de cada una de las gaussianas que conforman la densidad de probabilidad sobre cada estado.

A partir de ahora todo el desarrollo que sigue estará basado en modelos discretos, aunque la extensión al caso continuo resultaría sencilla.

B.2. Aplicaciones básicas de los modelos de Markov

Una vez vista la forma de un modelo de Markov, conviene revisar cuales son las tres aplicaciones básicas que estos tienen. Estas son:

- Calcular de una manera eficiente, a partir de un modelo y de una secuencia de observaciones, cuál es la probabilidad de que dicha secuencia haya sido generada por ese modelo.
- Obtener a partir de un modelo y de una secuencia de observaciones cuál es la secuencia de estados más *significativa* que describe a las observaciones.
- Ajustar los parámetros del modelo (λ) para maximizar las probabilidades $P(O|\lambda)$ de una serie de secuencias de observaciones O .

Para el caso que aquí se trata sólo son interesantes la primera y la última aplicación, evaluación y aprendizaje respectivamente. A continuación se estudiará el problema de evaluación. Un desarrollo más profundo de esta aplicación y de las otras dos puede recurrirse al tutorial sobre modelos ocultos de Markov de Rabiner ([Rab89]).

B.2.1. Problema de evaluación en HMM

Antes de comenzar a desarrollar estas dos aplicaciones será necesario fijar cuál es la notación que será empleada. El modelo de Markov tendrá una serie de estados S_i , $i = 1, \dots, N$. Las transiciones entre el estado actual, q_t , y el siguiente q_{t+1} , estarán reguladas por las probabilidades de transición:

$$a_{ij} = P[q_{t+1} = S_j | q_t = S_i], \quad 1 \leq i, j \leq N \quad (\text{B.3})$$

En cada instante t , se emitirá un símbolo v_k , de entre los M posibles, en cada estado S_j de los N existentes:

$$b_j(k) = P[v_k \text{ en } t | q_t = S_j], \quad 1 \leq j \leq N; 1 \leq k \leq M \quad (\text{B.4})$$

B. MODELOS OCULTOS DE MARKOV

Por último, es necesario conocer en cuál de los estados comienza la secuencia oculta del modelo. Este proceso, que también puede considerarse aleatorio, está gobernado por las variables π_i , que indican la probabilidad de comenzar la secuencia en un estado en particular, S_i :

$$\pi_i = P[q_1 = S_i], \quad 1 \leq i \leq N \quad (\text{B.5})$$

La evaluación de un modelo de Markov busca averiguar cual es la probabilidad de que una determinada secuencia pueda haber sido generada por un modelo en concreto. Este modelo vendrá definido por el conjunto de sus probabilidades de transición (matriz A), sus probabilidades de emisión (matriz B) y el vector π . El conjunto de estos tres parámetros es habitualmente representado por λ , que se refiere a todos los parámetros libres del modelo de Markov.

Se parte de una secuencia de observaciones, de un número cualquiera de elementos, T .

$$O = O_1 O_2 \cdots O_T \quad (\text{B.6})$$

En el caso de que se conozca la secuencia de estados seguida en el proceso de generación, o que la misma sea fija, el procedimiento es sencillo: partiendo de la secuencia de estados Q , $Q = q_1 q_2 \cdots q_T$, se calcula la probabilidad de la secuencia con una secuencia de estados fija.

$$P(O|Q, \lambda) = \prod_{t=1}^T P(O_t|q_t, \lambda) \quad (\text{B.7})$$

En cada instante el modelo se encontrará en un estado q_t , en el que con una probabilidad $b_{q_t}(O_t)$ emitirá el símbolo O_t . Por lo tanto la anterior probabilidad puede ser escrita como:

$$P(O|Q, \lambda) = b_{q_1}(O_1) \cdot b_{q_2}(O_2) \cdots b_{q_T}(O_T) \quad (\text{B.8})$$

Ahora bien, si, como es el caso general, la secuencia de estados es desconocida, será necesario evaluar la probabilidad de una secuencia frente a otra. La probabilidad de que una secuencia Q en particular dado el modelo es:

$$P(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \cdots a_{q_{T-1} q_T} \quad (\text{B.9})$$

Así la probabilidad de que ocurran ambas cosas a la vez, tomar un determinado camino Q y en él emitir la secuencia de observaciones O , no será más que el producto de las expresiones (B.8) y (B.9):

$$P(O, Q|\lambda) = P(O|Q, \lambda)P(Q, \lambda) \quad (\text{B.10})$$

Para obtener la probabilidad de una secuencia, no hay más que sumar esta última expresión sobre todos los posibles caminos que puede seguir:

$$\begin{aligned} P(O|\lambda) &= \sum_{Q_1, \dots, Q_n} P(O|Q, \lambda)P(Q|\lambda) & (\text{B.11}) \\ &= \sum_{Q_1, \dots, Q_n} \pi_{q_1} b_{q_1}(O_1) a_{q_1, q_2} b_{q_2}(O_2) \cdots a_{q_{T-1}, q_T} b_{q_T}(O_T) & (\text{B.12}) \end{aligned}$$

El problema que plantea la anterior expresión, como puede suponerse, es que cuando el tamaño del modelo comience a crecer es inabordable, computacionalmente hablando. Un camino alternativo para resolver este cálculo de manera eficiente es emplear el algoritmo *forward-backward*, que se presenta a continuación.

Inicialmente se parte de la variable hacia adelante $\alpha_t(i)$. Esta representa la probabilidad de la observación parcial de la secuencia hasta el instante t y el estado S_i en dicho instante, dado el modelo λ .

$$\alpha_t(i) = P(O_1, O_2, \dots, O_t, q_t = S_i|\lambda) \quad (\text{B.13})$$

Podemos ir calculando, de forma iterativa, estas variables siguiendo un procedimiento sencillo que consta de tres pasos:

1. Inicialización:

$$\alpha_t(i) = \pi_i b_i(O_1), \quad 1 \leq i \leq N \quad (\text{B.14})$$

2. Inducción:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^N \alpha_t(i) a_{ij} \right] b_j(O_{t+1}), \quad 1 \leq t \leq T-1, \quad 1 \leq j \leq N \quad (\text{B.15})$$

3. Finalización:

$$P(O|\lambda) = \sum_{i=1}^N \alpha_T(i) \quad (\text{B.16})$$

Así es posible calcular de una forma eficiente la probabilidad de que una secuencia haya sido generada por el modelo. De forma análoga, el cálculo podría haberse hecho partiendo del final de la secuencia. En este caso se hubiera empleado la variable hacia atrás $\beta_t(i)$:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \dots, O_T | q_t = S_i, \lambda) \quad (\text{B.17})$$

Este procedimiento, por simetría al anterior, puede resumirse en estos dos pasos:

1. Inicialización:

$$\beta_t(i) = 1, \quad 1 \leq i \leq N \quad (\text{B.18})$$

2. Inducción:

$$\beta_t(j) = \sum_{i=1}^N a_{ij} b_j(O_{t+1}) \beta_{t+1}(i), \quad t = T-1, T-2, \dots, 1, 1 \leq j \leq N \quad (\text{B.19})$$

B.3. Otros parámetros de los HMM útiles para calcular el kernel de Fisher

Durante el desarrollo del kernel de Fisher (sección 2.5) se ha visto que para la extracción del vector de puntuaciones de Fisher, es necesario conocer cuál es la esperanza de haber estado en un estado S_j y en él emitir el símbolo v_k . El cálculo de esta esperanza puede realizarse fácilmente a partir de los parámetros α y β comentados anteriormente, como se presenta a continuación.

La magnitud $\gamma_t(j)$ indica cuál es la probabilidad de encontrarse en un estado en particular (S_j) en un instante t a la vista de la secuencia y el modelo:

$$\gamma_t(j) = P(q_t = S_j | O, \lambda) \quad (\text{B.20})$$

Esta magnitud puede expresarse en términos de las variables α y β de la siguiente forma:

$$\gamma_t(j) = \frac{\alpha_t(j) \beta_t(j)}{P(O | \lambda)} = \frac{\alpha_t(j) \beta_t(j)}{\sum_{i=1}^N \alpha_t(i) \beta_t(i)} \quad (\text{B.21})$$

B.3. Otros parámetros de los HMM útiles para calcular el kernel de Fisher

donde $\alpha_t(j)$ calcula la secuencia parcial de observaciones $O_1 O_2 \dots O_t$ terminando en el estado S_j en instante t . Mientras, $\beta_t(j)$ se encarga de $O_{t+1} O_{t+2} \dots O_T$ hasta el estado S_j en el instante t . El factor de normalización $P(O|\lambda) = \sum_{j=1}^N \alpha_t(j) \beta_t(j)$ hace de $\gamma_t(j)$ una probabilidad, que de esta forma cumple que:

$$\sum_{j=1}^N \gamma_t(j) = 1 \quad (\text{B.22})$$

Si sumamos estas probabilidades a lo largo de toda la secuencia se obtiene la esperanza de haber pasado por un determinado estado S_j durante el la generación de la secuencia.

$$\sum_{t=1}^T \gamma_t(j) = \text{número esperado de veces que está en el estado } j \quad (\text{B.23})$$

Si en este sumatorio solo se consideran aquellos instantes en los que se emite un símbolo en particular, tendremos la esperanza de haber pasado por un estado en concreto y emitir en él un símbolo en particular.

$$\sum_{t=1}^T \gamma_t(j) = \text{número esperado de veces en el estado } j \text{ con } v_k \quad (\text{B.24})$$

APÉNDICE C

Prestaciones del kernel de Fisher

A continuación pueden verse las 32 curvas ROC, para cada una de las familias usadas en el capítulo 4. Para obtenerlas se ha usado una máquina de vectores soporte con kernel gaussiano, usando los parámetros $\gamma = 0,0005$ y $C = 100$. Se muestran las curvas obtenidas tanto sobre el conjunto de entrenamiento como sobre el de verificación o test. También se muestra en la gráfica un punto que indica los resultados obtenidos con el método Blast.

C. PRESTACIONES DEL KERNEL DE FISHER

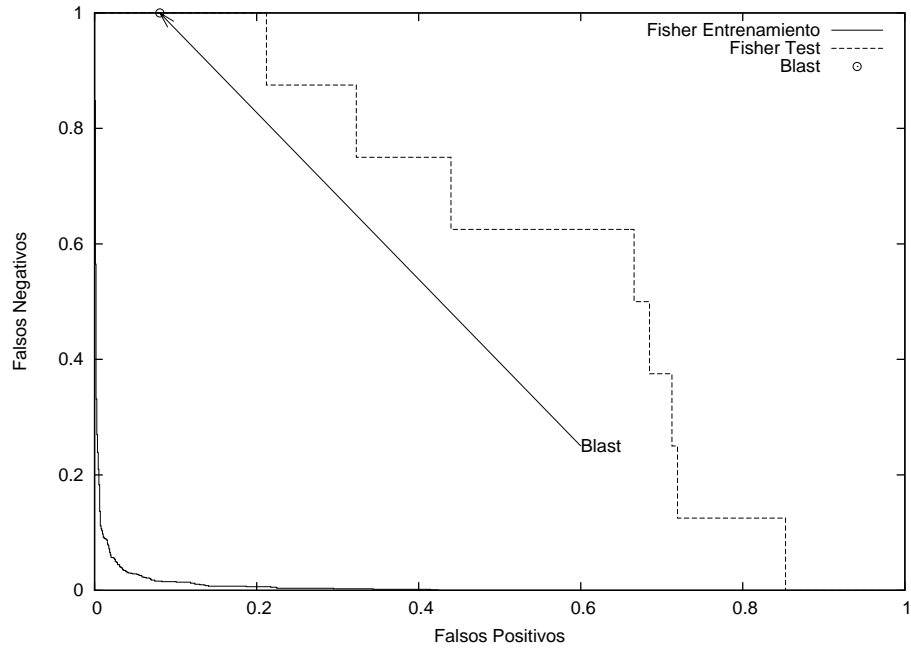


Figura C.1: Curva de prestaciones para la familia 1.1.1.2.

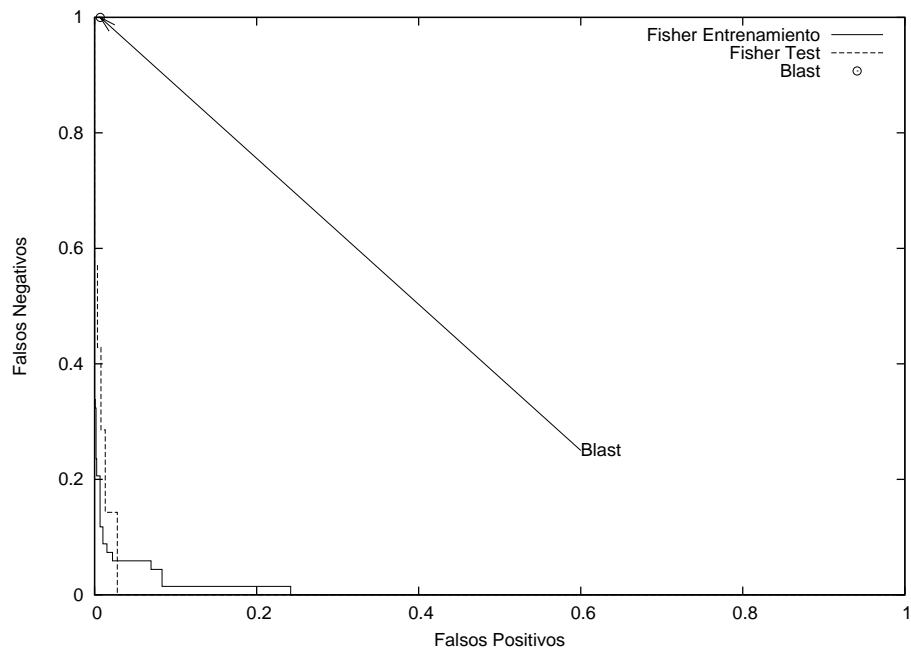


Figura C.2: Curva de prestaciones para la familia 1.25.1.1.

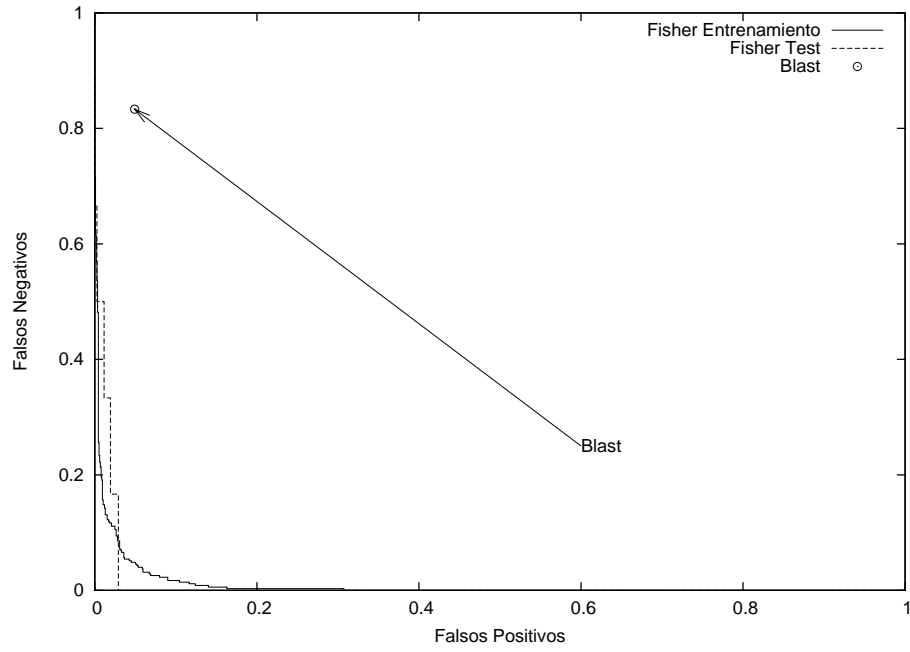


Figura C.3: Curva de prestaciones para la familia 1.25.1.2.

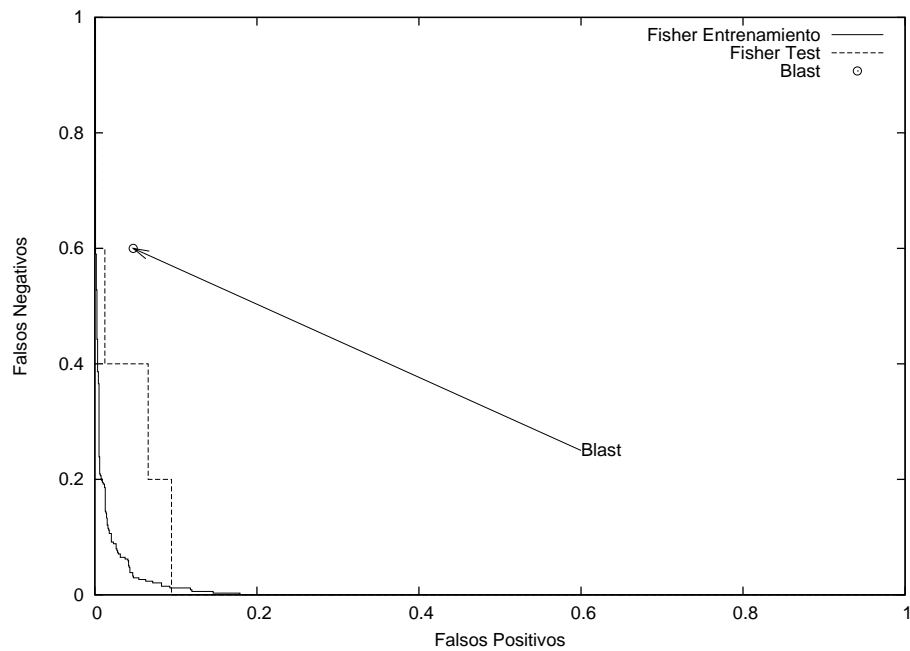


Figura C.4: Curva de prestaciones para la familia 1.25.1.3.

C. PRESTACIONES DEL KERNEL DE FISHER

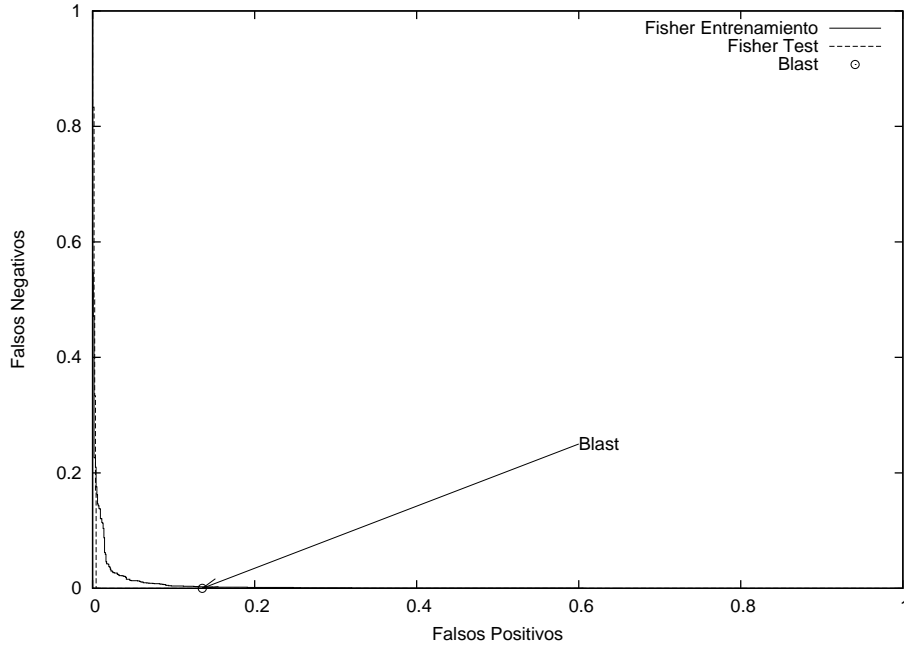


Figura C.5: Curva de prestaciones para la familia 1.34.1.4.

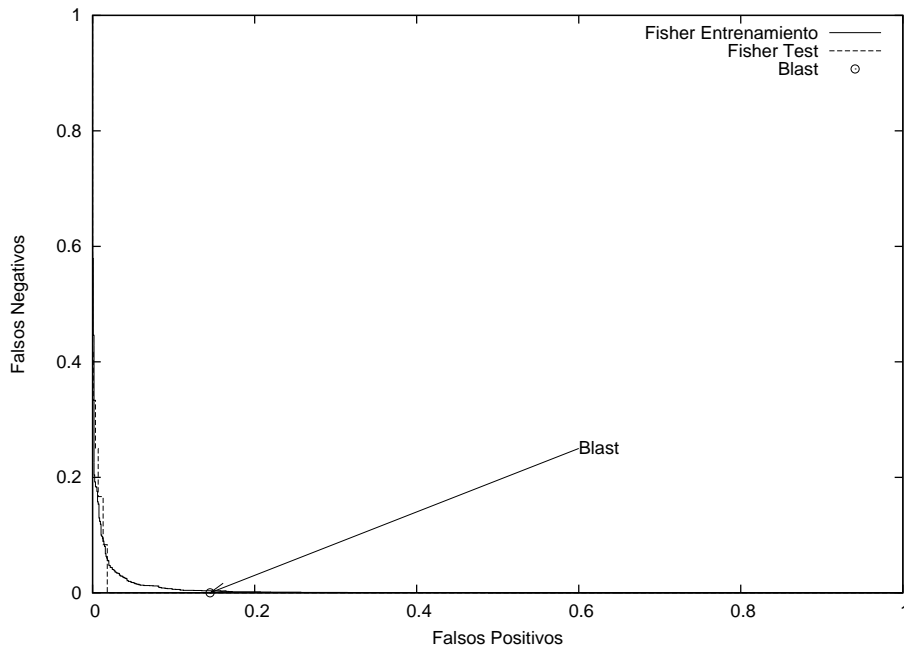


Figura C.6: Curva de prestaciones para la familia 1.34.1.5.

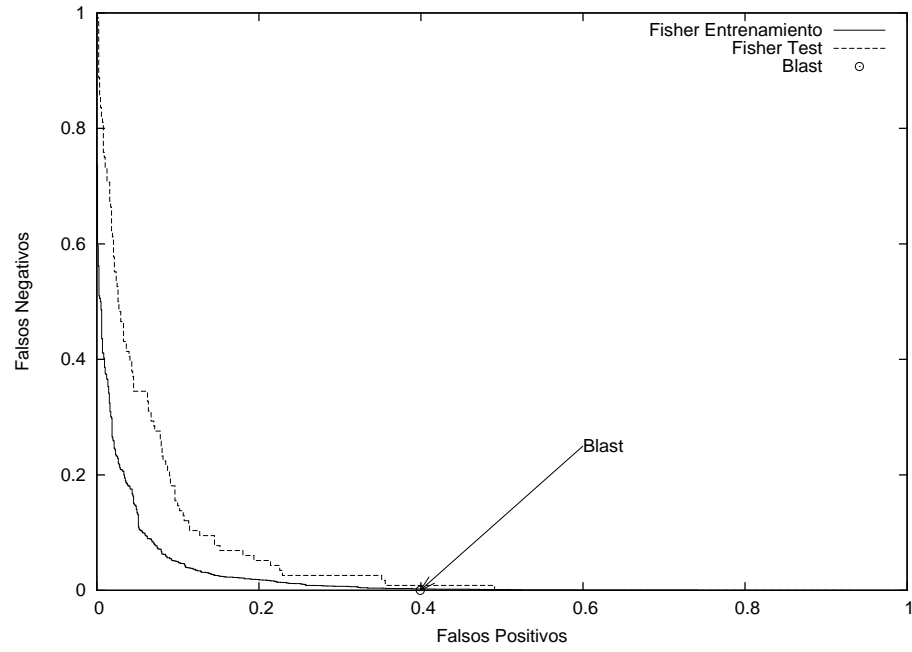


Figura C.7: Curva de prestaciones para la familia 2.1.1.1.

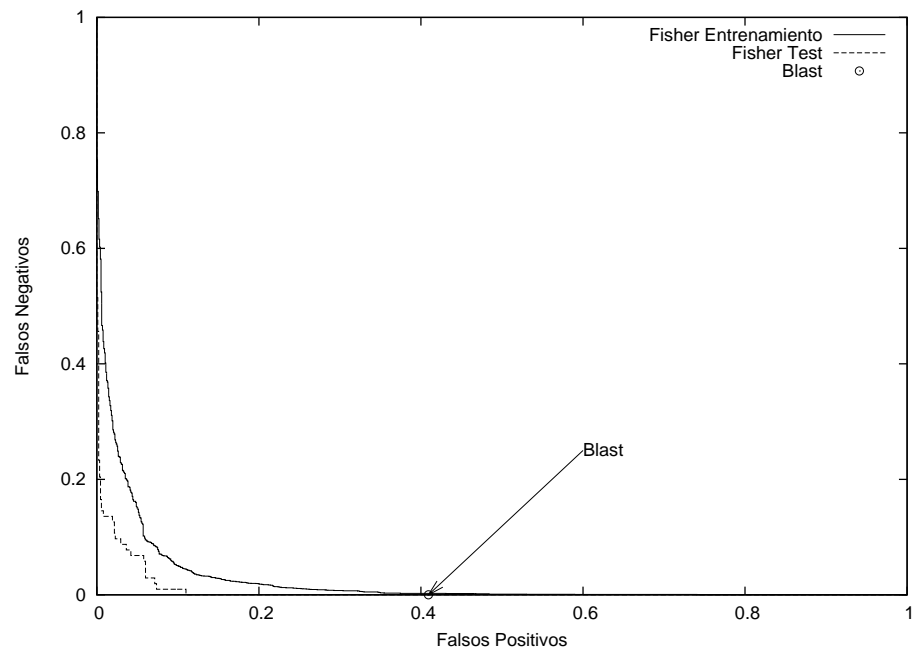


Figura C.8: Curva de prestaciones para la familia 2.1.1.2.

C. PRESTACIONES DEL KERNEL DE FISHER

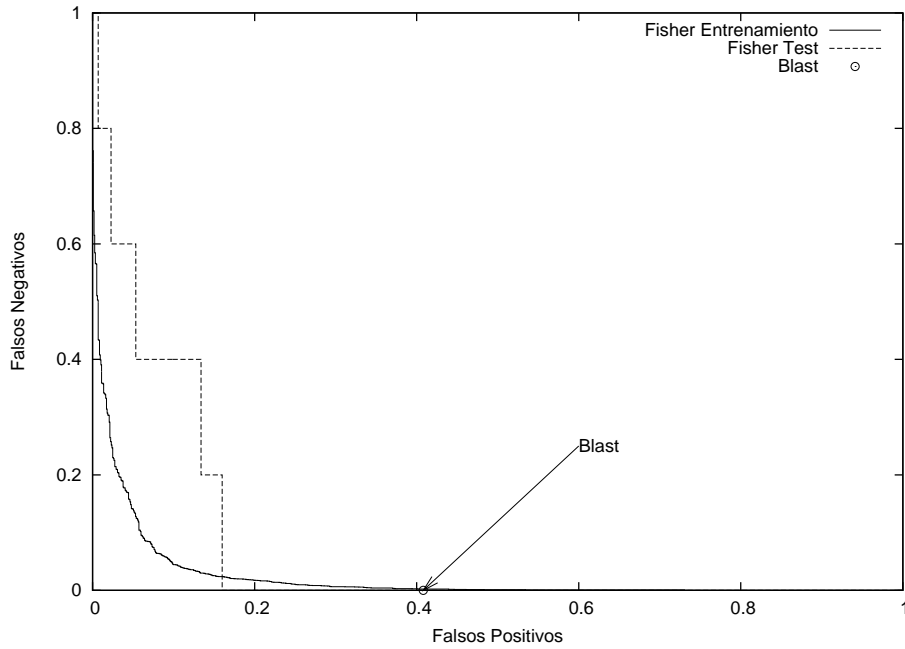


Figura C.9: Curva de prestaciones para la familia 2.1.1.3.

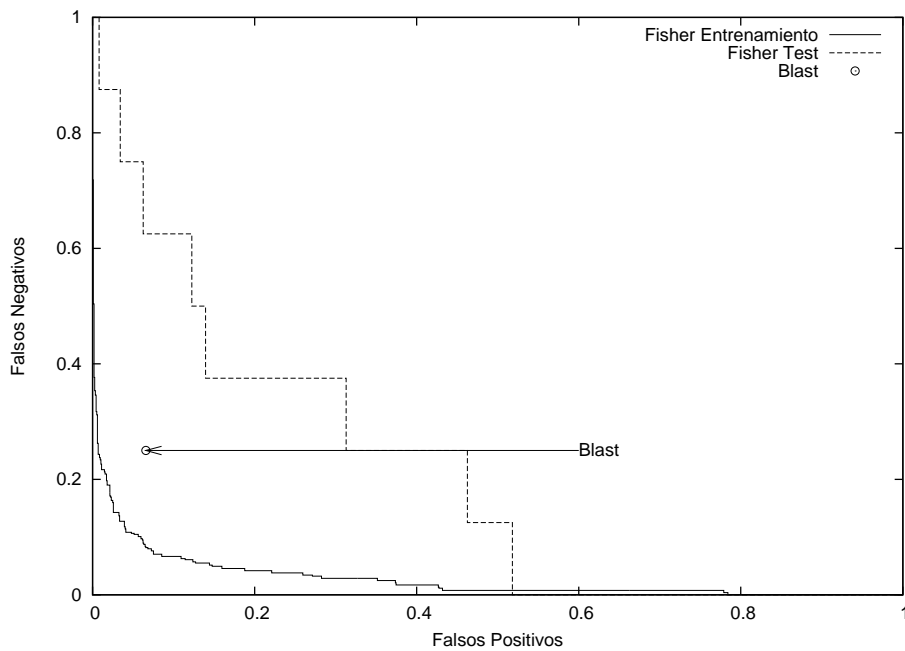


Figura C.10: Curva de prestaciones para la familia 2.1.1.4.

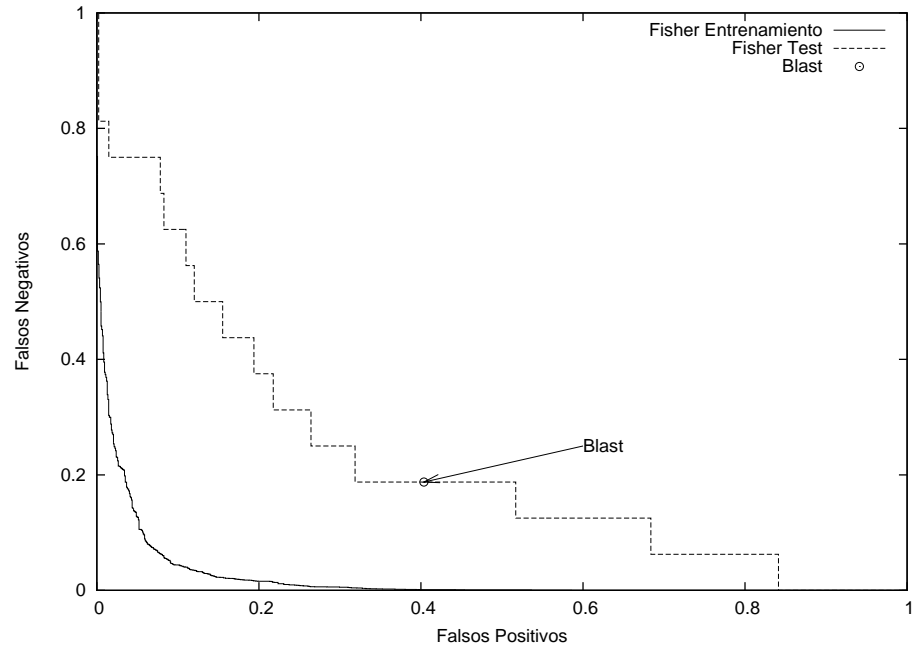


Figura C.11: Curva de prestaciones para la familia 2.1.1.5.

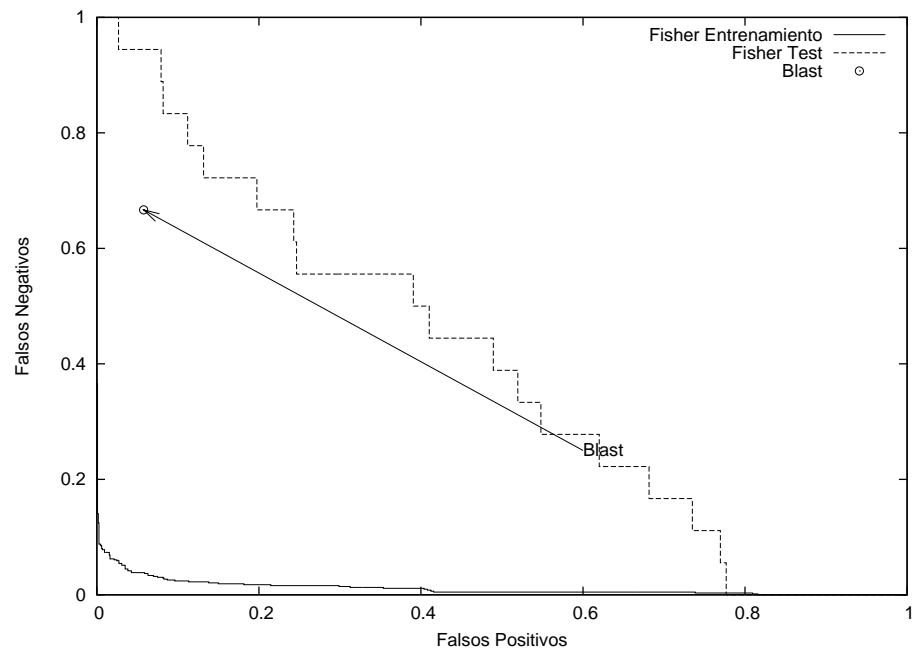


Figura C.12: Curva de prestaciones para la familia 2.19.1.1.

C. PRESTACIONES DEL KERNEL DE FISHER

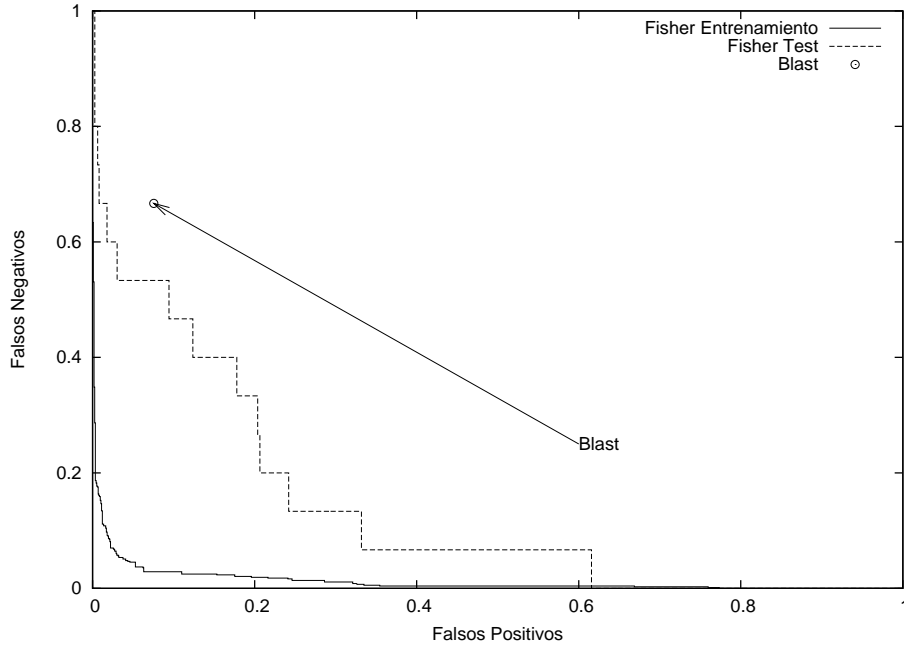


Figura C.13: Curva de prestaciones para la familia 2.31.1.1.

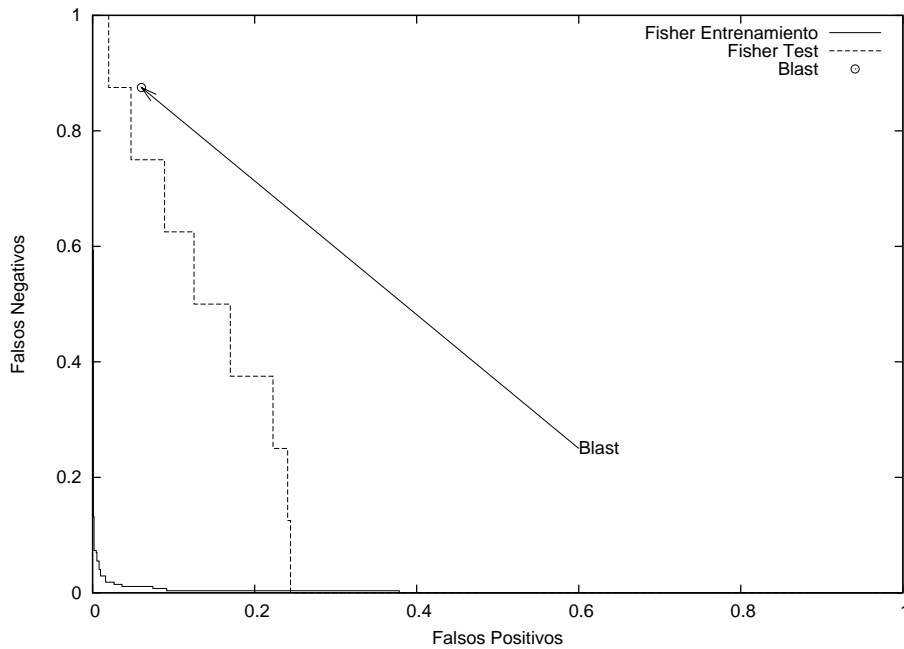


Figura C.14: Curva de prestaciones para la familia 2.31.1.2.

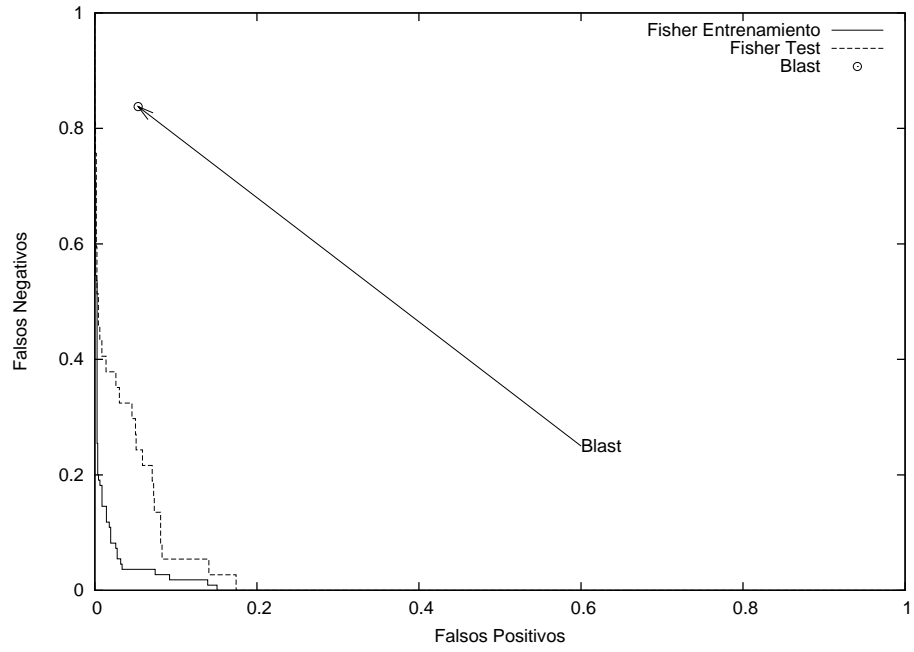


Figura C.15: Curva de prestaciones para la familia 2.34.1.1.

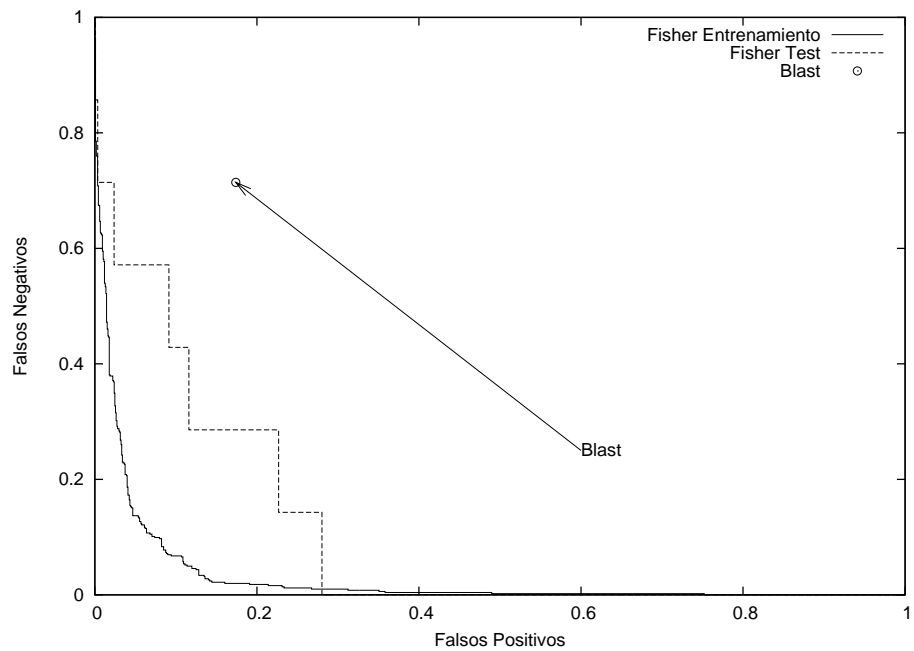


Figura C.16: Curva de prestaciones para la familia 2.41.1.1.

C. PRESTACIONES DEL KERNEL DE FISHER

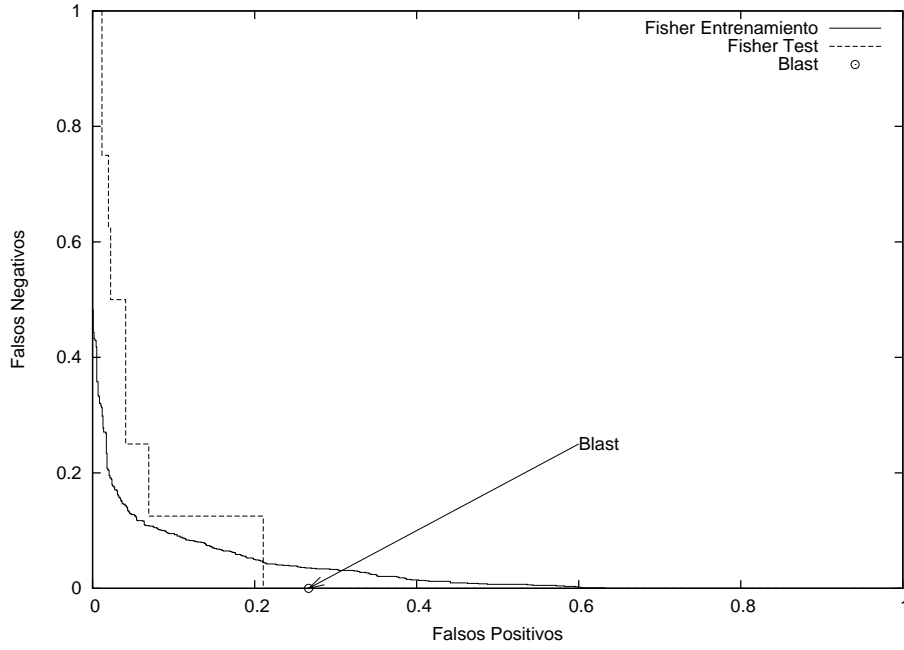


Figura C.17: Curva de prestaciones para la familia 2.5.1.1.

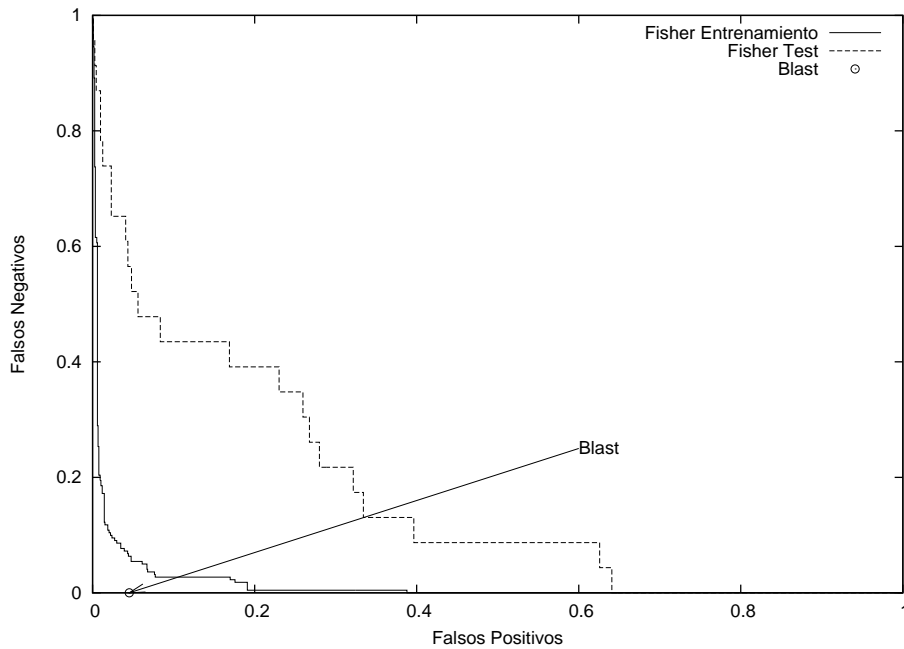


Figura C.18: Curva de prestaciones para la familia 2.5.1.3.

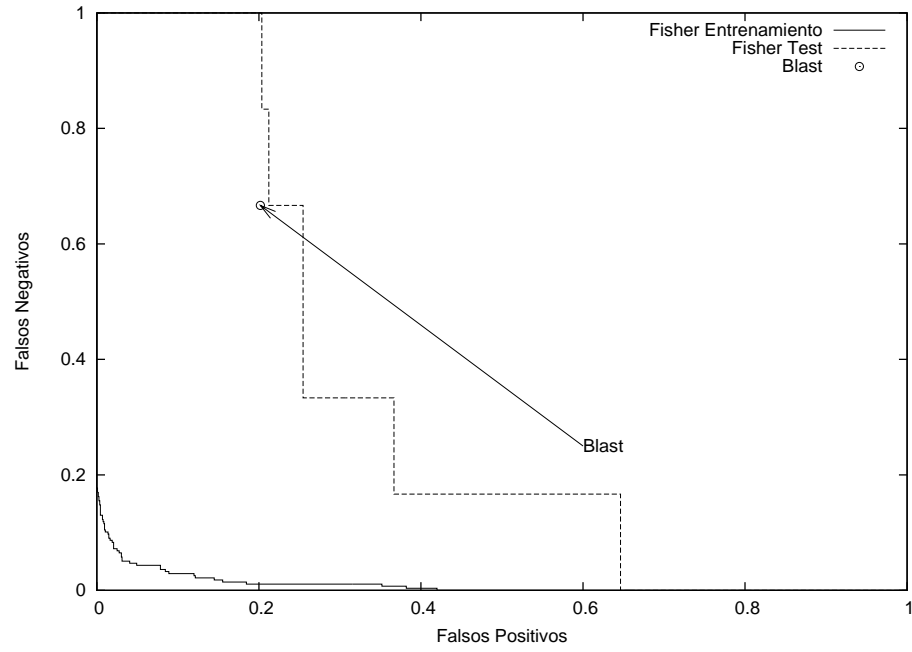


Figura C.19: Curva de prestaciones para la familia 2.8.1.2.

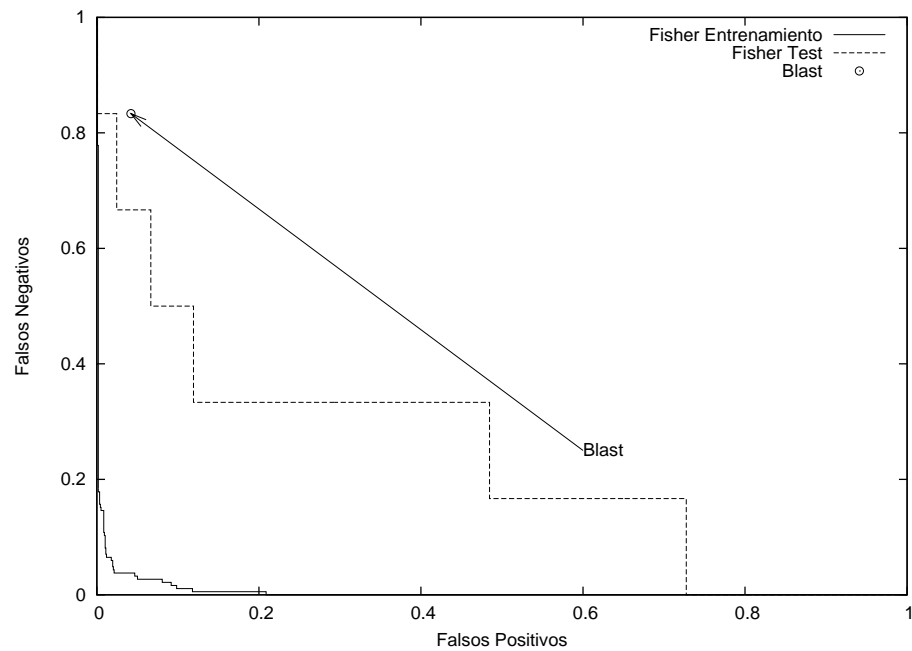


Figura C.20: Curva de prestaciones para la familia 2.8.1.4.

C. PRESTACIONES DEL KERNEL DE FISHER

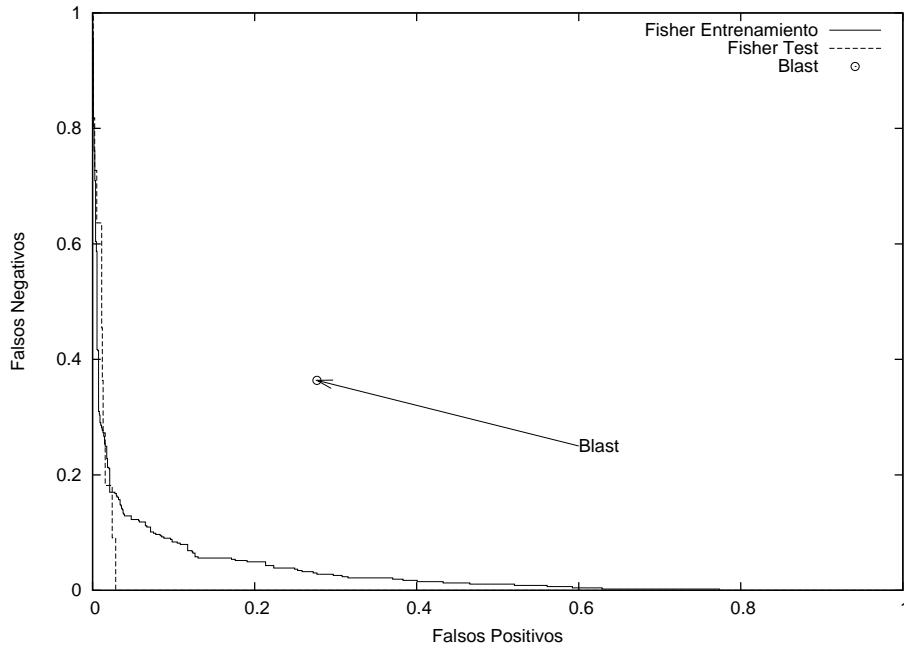


Figura C.21: Curva de prestaciones para la familia 3.1.1.1.

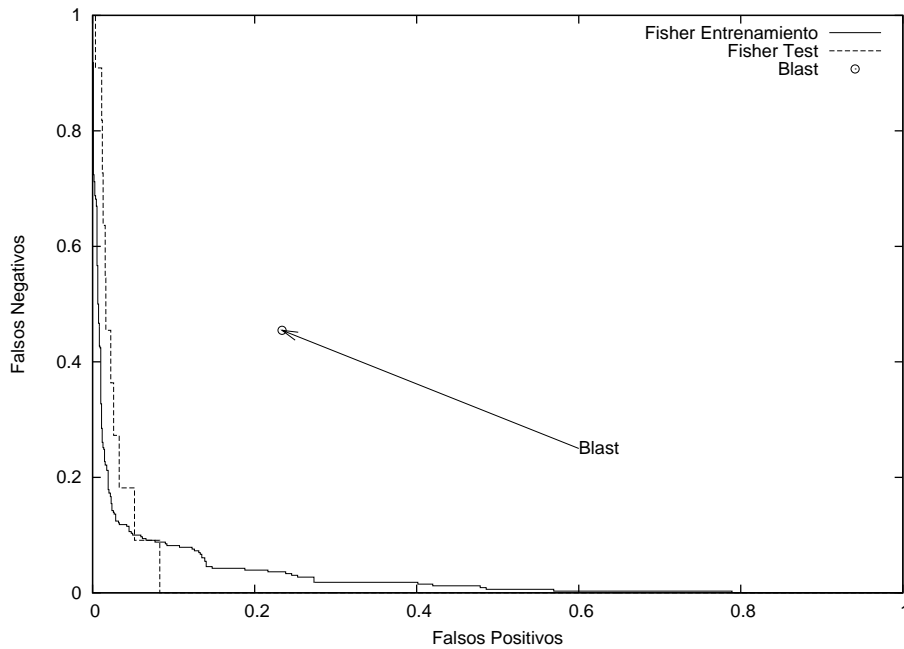


Figura C.22: Curva de prestaciones para la familia 3.1.1.3.

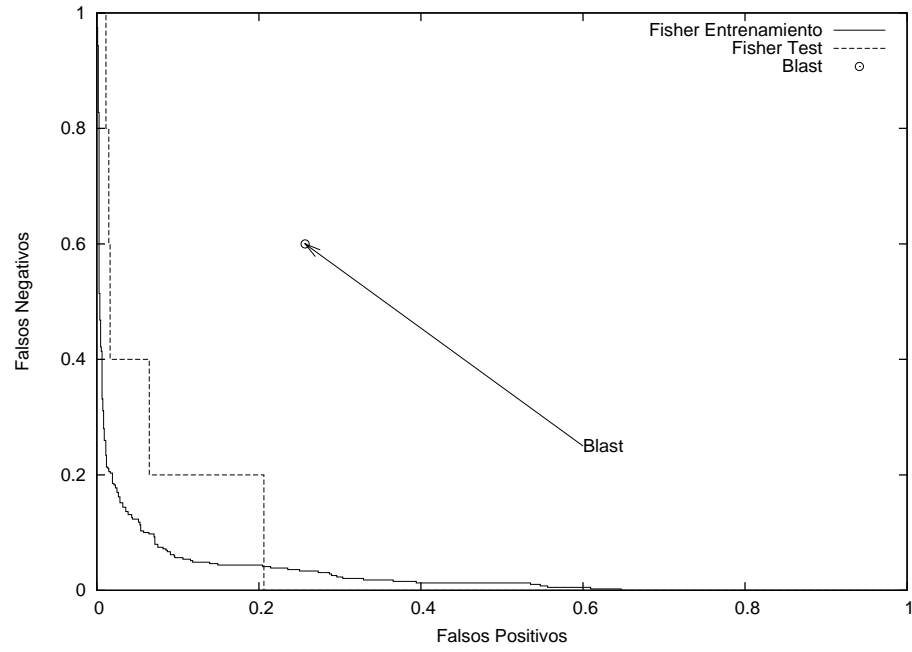


Figura C.23: Curva de prestaciones para la familia 3.1.1.5.

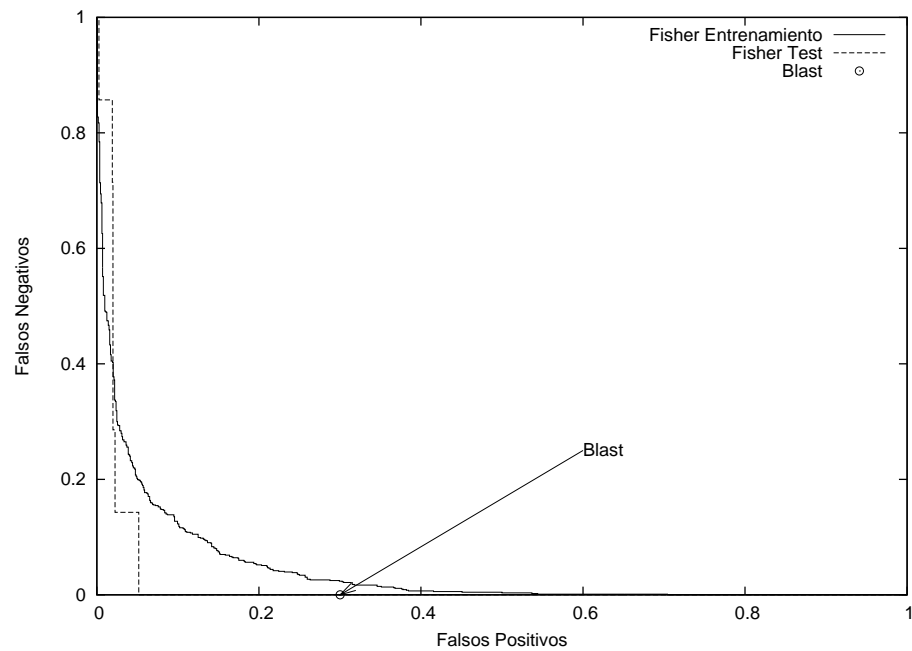


Figura C.24: Curva de prestaciones para la familia 3.19.1.1.

C. PRESTACIONES DEL KERNEL DE FISHER

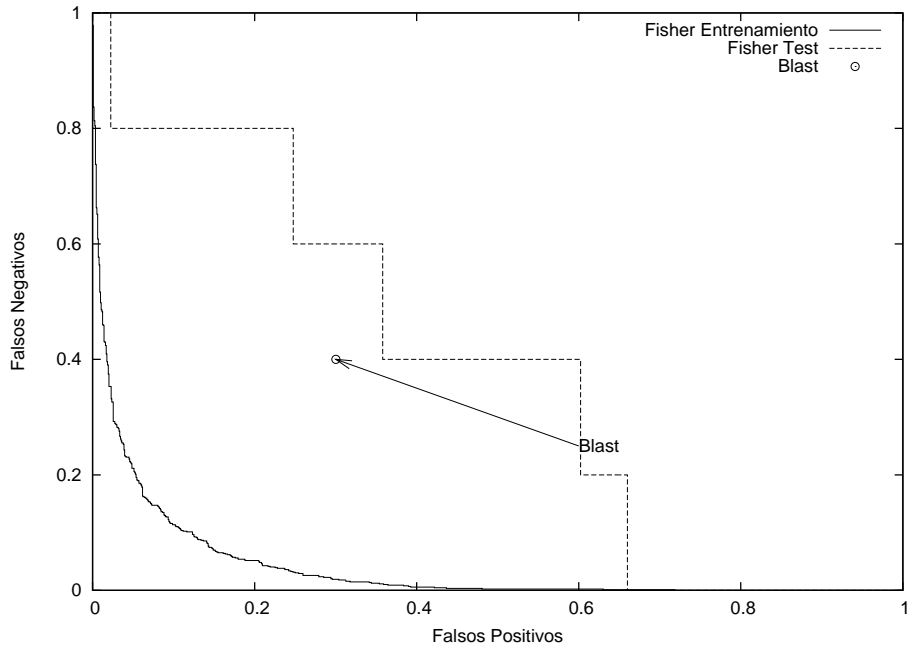


Figura C.25: Curva de prestaciones para la familia 3.19.1.4.

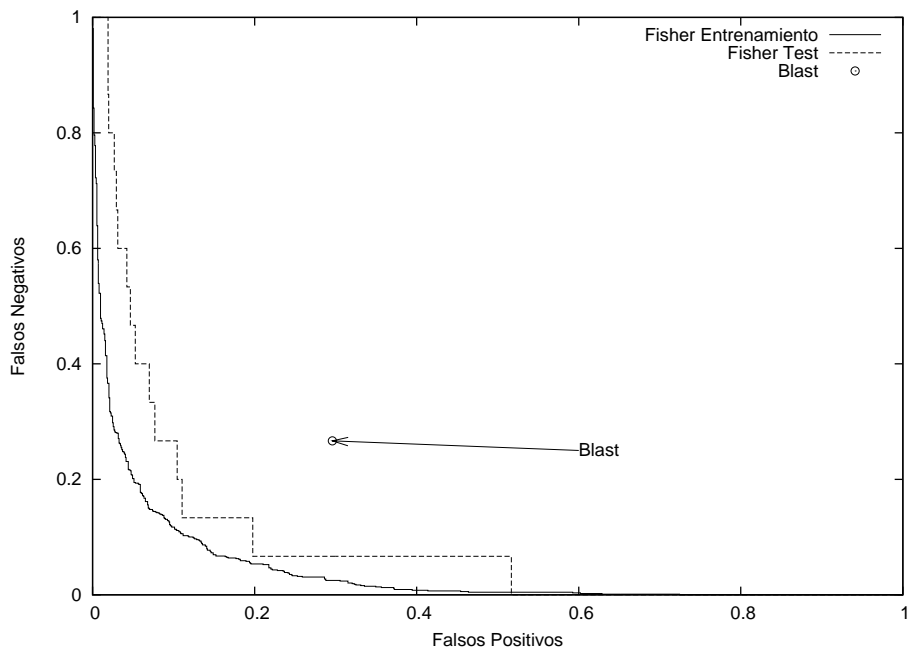


Figura C.26: Curva de prestaciones para la familia 3.19.1.5.

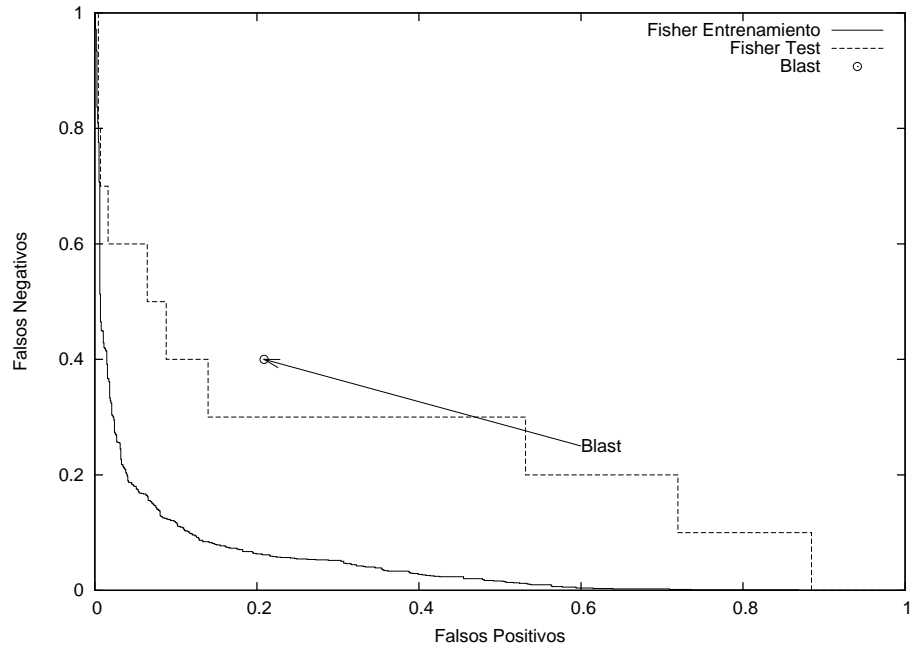


Figura C.27: Curva de prestaciones para la familia 3.25.1.1.

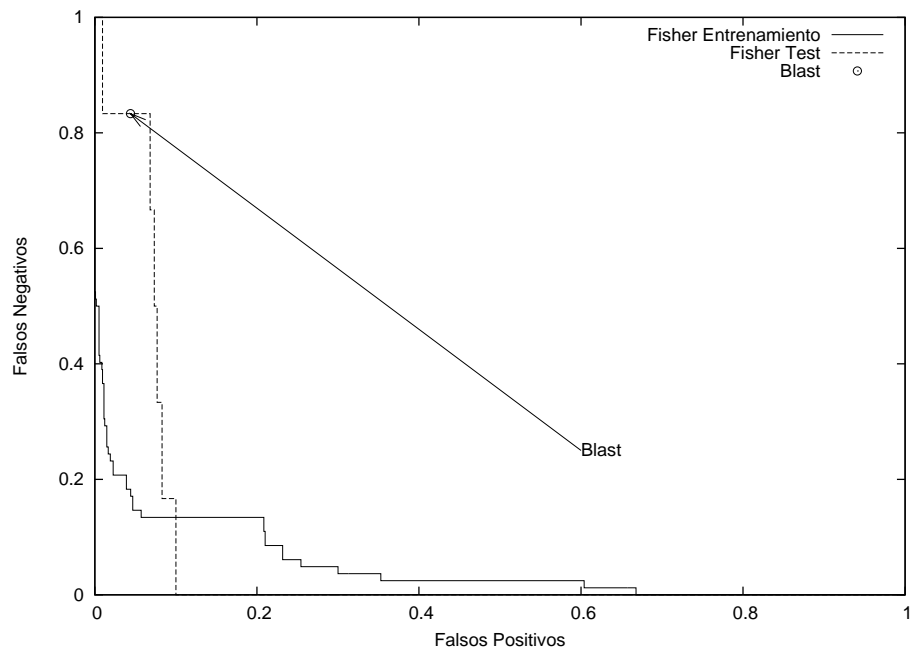


Figura C.28: Curva de prestaciones para la familia 3.25.1.3.

C. PRESTACIONES DEL KERNEL DE FISHER

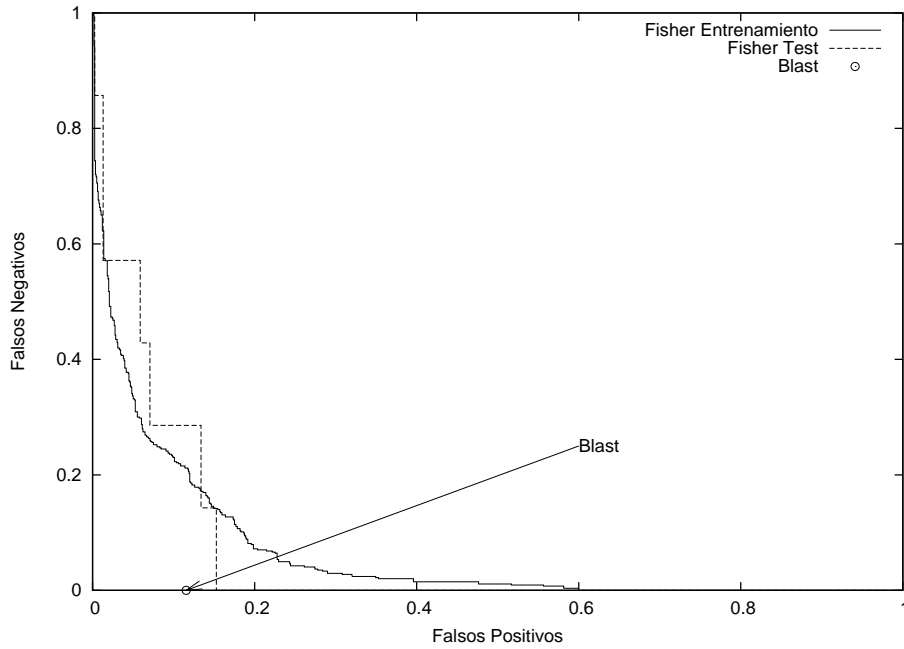


Figura C.29: Curva de prestaciones para la familia 3.33.1.1.

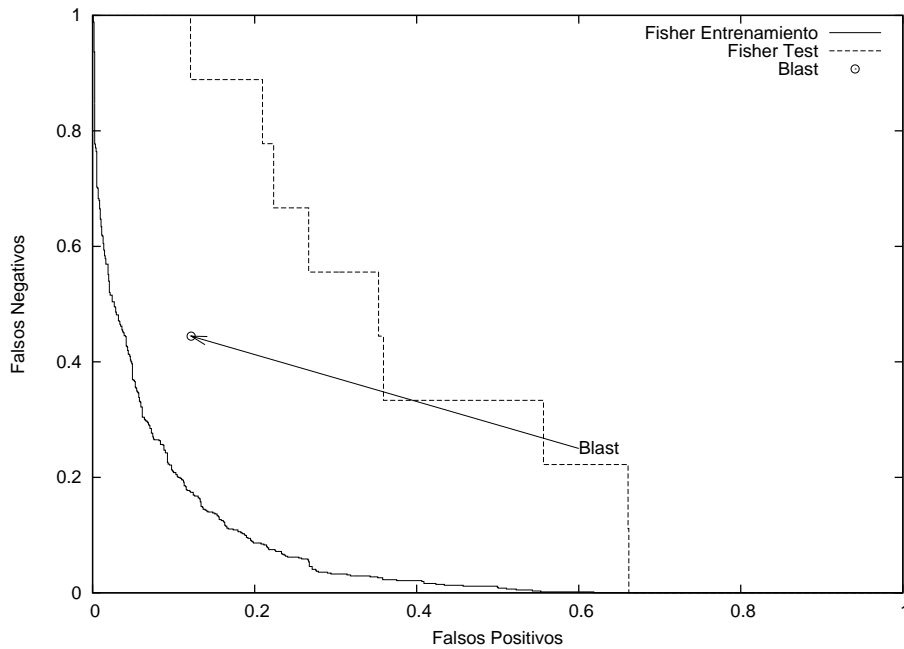


Figura C.30: Curva de prestaciones para la familia 3.33.1.5.

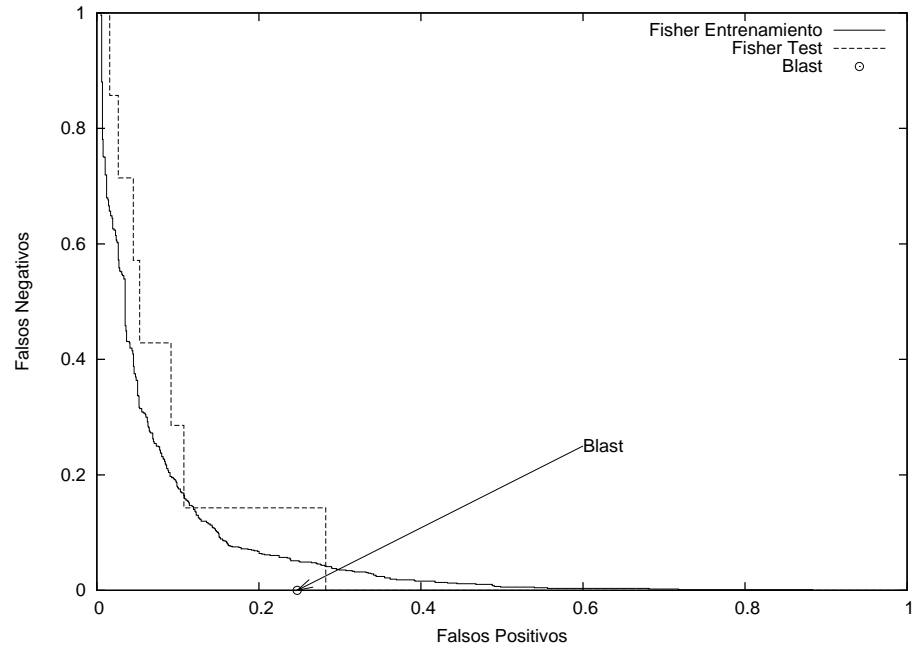


Figura C.31: Curva de prestaciones para la familia 3.50.1.7.

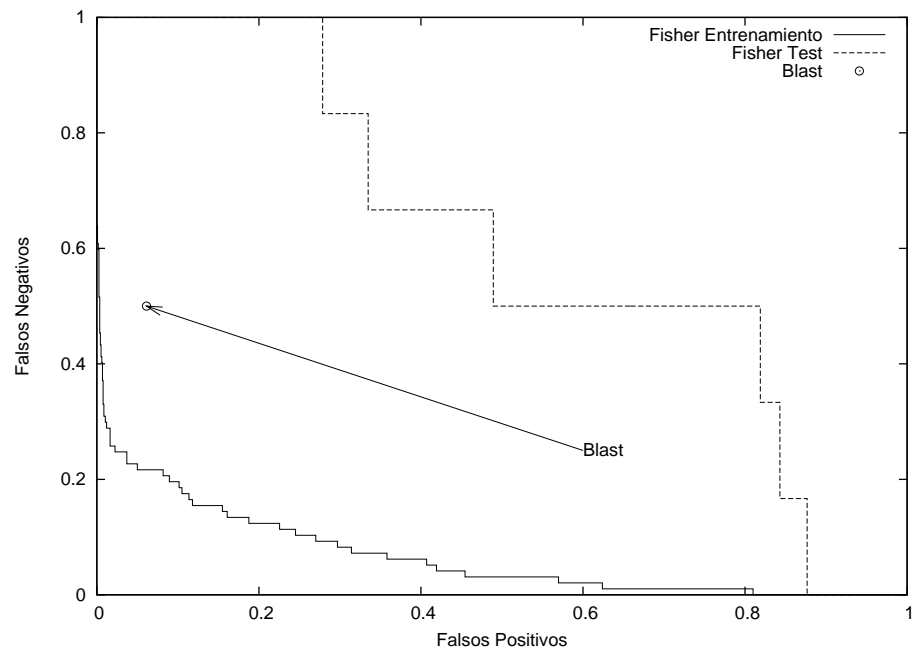


Figura C.32: Curva de prestaciones para la familia 3.73.1.2.

C. PRESTACIONES DEL KERNEL DE FISHER

BIBLIOGRAFÍA

- [AGM⁺90] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal in Molecular Biology*, 215:403–410, 1990.
- [Ama98] Shun Ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.
- [AMS⁺97] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped blast and psi-blast: A new generation of protein database search programs. *Nucl. Acids Res.*, 25:3389–3402, 1997.
- [BGV92] B. Boser, I. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory*, pages 144–152, 1992.
- [Bur98] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.
- [Chu89] G.A. Churchill. Stochastic models for heterogeneous dna sequences. *Bull. Math. Biol.*, 51:79–94, 1989.
- [CV95] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

BIBLIOGRAFÍA

- [DEKM98] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.
- [DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Recognition*. John Wesley & Sons, Inc., 2001.
- [Edd98a] S. Eddy. Hmmer user's guide: biological sequence analysis using prole hidden markov models, 1998.
- [Edd98b] S. R. Eddy. Profile hidden markov models (review). *Bioinformatics*, 14(9):755–763, 1998.
- [Fle87] R. Fletcher. *Practical methods of optimization*. John Wiley and Sons, Inc, 2 edition, 1987.
- [GME87] M. Gribskov, A.D. McLachlan, and D Eisenberg. Profile analysis: Detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA*, 84:4355–4358, 1987.
- [Hof00] Thomas Hofmann. Learning the similarity of documents: an information-geometric approach to document retrieval and categorization. *Advances in neural information processing systems*, 12:914–920, 2000.
- [JDH98] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies, 1998.
- [JDH99] T. Jaakkola, M. Diekhans, and D. Haussler. Using the fisher kernel method to detect remote protein homologies, 1999.
- [JH98] T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers, 1998.
- [Joa99] T. Joachims. Making large-scale svm learning practical. advances in kernel methods - support vector learning, b. schölkopf and c. burges and a. smola. *MIT-Press*, 1999.
- [KBH98] K. Karplus, C. Barrett, and R. Hughey. Hidden markov models for detecting remote protein homologies, 1998.

-
- [KBM⁺93] Anders Krogh, Michael Brown, I. Saira Mian, Kimmen Sjölander, and David Haussler. Hidden markov models in computational biology: Applications to protein modeling. Technical Report UCSC-CRL-93-32, 1993.
- [KL51] S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.
- [KSB⁺97] K. Karplus, Kimmen Sjölander, C. Barrett, M. Cline, D. Haussler, R. Hughey, L. Holm, and C. Sander. Predicting protein structure using hidden markov models. *Proteins: Structure, Function and Genetics*, 1:134–139, 1997.
- [LGG] Nicholas M. Luscombe, Dov Greenbaum, and Mark Gerstein. What is bioinformatics? an introduction and overview.
- [MBHC95] Alexey G. Murzin, Steven E. Brenner, Tim Hubbard, and Cyrus Chothia. Scop: A structural classification of proteins database for the investigation of sequences and structures. *Journal in Molecular Biology*, 1995.
- [NW70] S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal in Molecular Biology*, 40:443–453, 1970.
- [PL88] W. Pearson and D. Lipman. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA*, 85:2444–2448, 1988.
- [QB] Le Quan and Samy Bengio. Hybrid generative-discriminative models for speech and speaker recognition.
- [Rab89] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Febrero 1989.
- [SG02] N. Smith and M. Gales. Speech recognition using svms, 2002.
- [SSTV] Craig Saunders, John Shaw-Taylor, and Alexei Vinokourov. String kernels, fisher kernels and finite state automata.
-

BIBLIOGRAFÍA

- [Tay86] W.R. Taylor. Identification of protein sequence homology by consensus template alignment. *Journal in Molecular Biology*, 188:233–258, 1986.
- [Vap95] V. Vapnik. *The Nature of Statistical Learning Theory*. John Wiley and Sons, Inc, 1995.
- [Vap98] V. Vapnik. *Statistical learning theory*. New York Wiley, 1998.
- [Wat95] Michael S. Waterman. *Introduction to computational biology*. Chapman & Hall, 1995.
- [YEK⁺01] Steve Young, Gunnar Evermann, Dan Kershaw, Gareth Moore an Julian Odell, Dave Ollason, Valtcho Valtchev, and Phil Woodland. The htk book. <http://htk.eng.cam.ac.uk/docs/docs.shtml>, 2001.

Parte II
Presupuesto

Presupuesto

A continuación se presenta el presupuesto de la realización del presente proyecto de fin de carrera. El presupuesto está dividido en tres partes: una lista de tareas realizadas y una lista de los materiales que se han sido utilizados en la realización de las mismas y la lista del personal involucrado en el proyecto. Todo ello se une en un resumen en el que se contabilizan los gastos totales acumulados durante el periodo de desarrollo del proyecto.

Tareas realizadas

Para la realización de este proyecto se han tenido que abordar y superar una serie de tareas que se enumeran en la siguiente lista:

- Estudio de las nuevas técnicas de clasificación con máquinas de vectores soporte.
- Recopilación de las principales tareas dentro del mundo de la bioinformática, prestando especial atención a las problemas en los que intervienen secuencias.
- Búsqueda de información, revisión y estudio sobre el kernel de Fisher.
- Implementación del kernel de Fisher y de la plataforma de simulación.

-
- Recopilación de bases de datos para las simulaciones.
 - Realización de las simulaciones y recopilación de resultados.
 - Redacción del presente informe.

Estas tareas se han realizado desde el mes de septiembre de 2001 hasta noviembre de 2002, un total de 15 meses.

Coste de material

El material empleado en la realización del proyecto consta de los siguientes elementos:

- Lugar de trabajo: Se dispone de una oficina alquilada con un coste aproximado de 800 € al mes, incluyendo calefacción, baño, luz y agua corriente. El coste de limpieza suma otros 400 € mensuales. El despacho es compartido por 8 personas, por lo que sólo se contabilizará la parte proporcional del coste total: 150 € al mes. Durante los 15 meses de proyecto el coste acumulado es de 2.250 €.
- Material de oficina: Dentro de este concepto se incluye todo el material desechable empleado en el proyecto, impresión de artículos, hojas, carpetas, etc. El total estimado es de 60 €.
- Equipo informático: Se ha empleado un ordenador personal cuyo valor aproximando es de 1.200 €. Aunque el equipo se ha empleado exclusivamente en este proyecto, tras la finalización del mismo se empleará en otros, por lo que en concepto de amortización del material solo se considerará el 80 % de su precio original. Se acompaña con una impresora valorada en 500 €, a dividir entre las 8 personas de la oficina, que junto a la misma deducción por amortización que en el caso anterior, tiene un coste de 50 €. El total invertido en equipo informático es de 1.010 €.
- Coste de licencias software: Tanto el sistema operativo como el software empleado en la realización del proyecto es de licencia pública y se encuentra disponible en la web, por lo que el coste asociado a este concepto es de 0 €. El software utilizado ha sido el siguiente: sistema operativo Debian GNU/Linux 3.0, Python 2.2, Numeric Python 22.0, Biopython 1.00 (*), Octave 2.1, HTK 3.1.1 (*), LibSVM 2.36

(*), SVM-light 5.00 (*), Blast 1.1, HMMER 2.2g, Rasmol 2.6, Gnu-plot 3.7.2, TeTeX 1.0, Mozilla 1.0 y Emacs 21. Todo este software es de libre distribución y la mayor parte de él forma parte de la distribución de Debian de Linux empleada. El resto, marcados con un *, puede encontrarse en diferentes direcciones de Internet.

- Conexión a Internet: La tarifa plana ADSL tiene un coste mensual de 42 €. A lo largo de 15 meses, el coste será de 630 €.
- Impresión, encuadernación y copias del informe: El coste de la impresión y encuadernación de las copias necesarias del proyecto asciende a 100 €.

Costes del personal

Las tareas anteriormente citadas han sido realizadas por una única persona desde el mes de septiembre del año 2001. La dedicación al proyecto ha sido de 3 horas al día durante los primeros ocho meses y de jornada completa (8 horas) desde junio del 2002 hasta noviembre del mismo año. Teniendo en cuenta el mes de vacaciones del año 2001 y el del 2002, y computando cada mes como 20 días laborables, el número total de horas empleadas en la realización del proyecto ha sido de 1.220. Según los cánones establecidos, el coste de una hora de trabajo de un Ingeniero Superior de Telecomunicación es de 60 €, por lo que computando el sueldo total del encargado del proyecto asciende a 73.200 €.

El salario del director del proyecto se estima, de forma general, como un 7 % del coste de material del proyecto. Dado que el coste del material es de 4.050 €, el salario del director es de 283,50 €.

Por último, hay que tener en cuenta los gastos sociales. Estos suponen un 35 % del coste del personal involucrado en el proyecto, 25.719,23 €.

Costes totales

Finalmente, el resumen de los costes en los que se ha incurrido durante la realización del proyecto, teniendo en cuenta la aplicación de 16 % de impuestos, es el siguiente:

-
- **Costes de material:**
 - Lugar de trabajo: 2.250 €.
 - Material de oficina: 60 €.
 - Equipo informático: 1.010 €.
 - Coste de licencias software: 0 €.
 - Conexión a Internet: 630 €.
 - Impresión, encuadernación y copias del informe: 100 €.
 - Subtotal: 4.050 €.

 - **Costes de personal:**
 - Ingeniero encargado del proyecto: 73.200 €.
 - Director del proyecto: 283,50 €.
 - Gastos sociales: 25.719,23 €.
 - Subtotal: 99.202,73 €.

 - **Total gastos:** 103.252,73 €.

 - **Impuestos (16%):** 16.520,44 €.

 - **Total:** 119.773,17 €.

El coste total del proyecto asciende a *ciento diecinueve mil setecientos setenta y tres euro con diecisiete céntimos*.

Leganés, Noviembre de 2002.

Fdo. Pablo Barrera González.
Ingeniero de Telecomunicación.