Universidad Nacional de Educación a Distancia

Facultad de Ciencias

Máster en Matemáticas Avanzadas. Curso 2009/2010

Trabajo de Fin de Máster

Aprendizaje Estadístico con Funciones Kernel

Tutor: Prof. Hilario Navarro Veguillas

Autor: Juan José Gibaja Martíns

Septiembre de 2010

Índice

Presentación

El presente documento constituye la memoria del *Trabajo de Fin de Máster* del *Máster en Matemáticas Avanzadas* de la Facultad de Ciencias de la UNED y su título es "Aprendizaje Estadístico con Funciones *Kernel*".

La relevancia del tema es incuestionable. Por una parte, vivimos en una sociedad crecientemente interconectada, en la que los ciudadanos generamos de forma continua flujos de datos que se registran y analizan prácticamente *online* y con los fines más variados -comerciales, médicos, legales,...- (The Economist, 2010; Baker, 2008). Por otra parte, y simultáneamente, se está produciendo un cambio de paradigma en el mundo científico, que otorga cada vez una menor importancia a los modelos y un mayor peso a los datos. El paradigma dominante en las últimas décadas, denominado *theory-driven approach*, se está viendo reemplazado por el llamado *data-driven approach* (Anderson, 2008; Hey *et al.*, 2009; Breiman, 2001).

Dada la creciente importancia y la omnipresencia de los datos, se antoja imprescindible contar con herramientas y procedimientos de análisis y detección de patrones que faciliten la extracción de oro de entre esas toneladas de mineral que se generan de forma continua. Estas herramientas y procedimientos están, no obstante, sometidos a un elevado nivel de exigencia ya que la tarea a la que se enfrentan no es sencilla. Entre los requisitos que deben cumplir estos procedimientos cabe destacar dos que, lamentablemente, son contrapuestos: deben ser capaces de detectar relaciones complejas y deben ser eficientes:

- En cuanto a la capacidad para detectar relaciones complejas, se trata de una exigencia básica para un algoritmo de detección de patrones, ya que las pautas y regularidades que se presentan en la vida real no suelen ser, en muchos casos, sencillas.

- La eficiencia es otro aspecto básico. En efecto, la creciente facilidad para recoger y generar datos, por una parte, así como el descenso en el coste de su almacenamiento[1], por la otra, son dos factores que explican el enorme tamaño de las bases de datos en las que se pretende explorar la existencia de pautas o patrones. Ya no sirve con que un algoritmo funcione bien cuando se enfrenta a unos cientos de registros; los procedimientos de detección de pautas deben ser capaces de dar respuestas en tiempos razonables ante conjuntos de datos de cientos de miles de registros.

Así las cosas, el dilema está servido:

- Los algoritmos lineales, que son los que se desarrollaron en primer lugar, funcionan razonablemente bien con grandes bases de datos (es decir, son eficientes) pero son incapaces de detectar relaciones o pautas complejas.

- Por su parte, los procedimientos no lineales presentan las características opuestas: permiten la detección de pautas o patrones muy complejos pero no resultan eficientes.

Las funciones *kernel* aparecen como una posible (y muy prometedora) solución a este dilema ya

1 El coste de almacenamiento (medido como cociente entre el precio de venta de una unidad de disco duro y su capacidad) era en 1956 de 10.000 dólares/Mbyte. En la actualidad (agosto de 2010) este ratio se sitúa aproximadamente en 1 centavo de dólar/Mbyte [http://ns1758.ca/winch/winchest.html]

que, como se pondrá de manifiesto, permiten conjugar la eficiencia de los algoritmos lineales con la flexibilidad de los no lineales diluyendo, de este modo, el dilema planteado. La idea básica no es nueva sino que data de comienzos del siglo XX (Mercer, 1909). Sin embargo, el desarrollo teórico preciso no tuvo lugar hasta comienzos de la década de los noventa (Boser, Guyon y Vapnik, 1992), con la aparición de las llamadas *support vector machines* (máquinas de vector soporte). Algunos nombres propios en el desarrollo de esta teoría son los de Vapnik, Schölkopf, Mercer, Aizerman o Aronszajn.

La aparición, y rápida adopción por parte de la comunidad científica, de las *support vector machines* supuso el pistoletazo de salida para una serie de aportaciones centradas en la reformulación y adaptación de las técnicas clásicas del análisis de datos. El propósito de dichas reformulaciones era permitir el empleo de las mencionadas técnicas clásicas dentro del nuevo marco proporcionado por las funciones *kernel*.

Precisamente, es en el ámbito de las funciones *kernel* donde se enmarca el presente *Trabajo de Fin de Máster.* En concreto, en lo que sigue se tratarán los siguientes temas:

- Se expondrá el cambio de paradigma que se está produciendo en el ámbito de la ciencia, que está desplazando el centro de gravedad de los modelos a los datos. En esta línea, se destacará el papel que, en este nuevo paradigma, jugará el desarrollo de algoritmos de detección de patrones.

- Se describirán los objetivos básicos de los algoritmos de detección de patrones así como de la disciplina del aprendizaje estadístico. Además, se pondrán de manifiesto las ventajas e inconvenientes de las técnicas lineales y no lineales de detección de patrones, tanto en el ámbito supervisado como en el no supervisado.

- Se expondrá la solución *kernel* y se analizará su adecuación como vía para tratar de reconciliar el dilema entre lo lineal y lo no lineal. En este documento se propondrá la solución *kernel* como una inteligente vía para integrar lo mejor de estos dos mundos.

- Elegida la solución *kernel*, se efectuará una recopilación de la bibliografía más relevante en este ámbito y, muy en particular, en el campo de las *support vector machines*, que constituyen la solución *kernel* más asentada y estudiada.

- Seguidamente se adaptarán algunos procedimientos tradicionales de detección de patrones para que sea posible su utilización con funciones *kernel.* En concreto, se adaptará un clasificador *hard* y un clasificador *soft* (dentro del aprendizaje de tipo supervisado) así como la conocida técnica del análisis de componentes principales (en el marco del aprendizaje de tipo no supervisado).

- Para cada uno de estos algoritmos adaptados se presentarán ejemplos de diverso tamaño.

- Para la resolución de los ejemplos se empleará el entorno de programación R y, en concreto, se hará uso de los *packages* kernlab, e1071 y FactoMineR.

Los temas se organizan del siguiente modo:

- El tema 1 es introductorio. En él se plantea el cambio de paradigma que se está produciendo

en el mundo científico, se establece el papel que en esta nueva situación debe jugar el aprendizaje estadístico, se muestran las ventajas e inconvenientes de los métodos lineales y no lineales y se esboza la solución *kernel* como vía de reconciliación de estas dos alternativas. Al final del tema se presenta una bibliografía básica sobre las funciones *kernel* en general y sobre las *support vector machines* en particular.

- En el tema 2, se expone con detalle el fundamento teórico de la solución *kernel*; se proporciona una caracterización de las funciones *kernel*; se muestra cómo construir nuevas funciones *kernel* a partir de otras y se presentan las características básicas de dos funciones *kernel* muy comunes: el *kernel polinómico* y el *kernel gaussiano*.

- A continuación, en el tercer tema, se muestra una serie de procedimientos básicos (algoritmos) que serán de utilidad en los temas sucesivos.

- El tema 4, se dedica a la aplicación de las funciones *kernel* al aprendizaje de tipo supervisado, lo que da como resultado las llamadas *support vector machines*. En este tema se presentan los clasificadores de margen máximo *hard* y *soft* en sus formulaciones primal y dual. Para ello es necesario introducir los conceptos de *problema de optimización lagrangiano* y *problema dual*. Finalmente, se adaptan ambos clasificadores para que puedan ser utilizados con funciones *kernel*. El tema finaliza con la resolución detallada de un ejercicio de ilustración mediante el empleo del entorno de programación R.

- El siguiente tema está dedicado a la aplicación de las funciones *kernel* al aprendizaje de tipo no supervisado y, en concreto, al análisis de componentes principales. Se presenta la formulación del análisis de componentes principales a partir de la descomposición espectral de la matriz de covarianzas y de la matriz de productos escalares entre las coordenadas de los individuos. Esta última formulación es la que permite la adaptación del procedimiento para que pueda ser empleado con funciones *kernel:* es el llamado *kernel PCA*. El tema finaliza con la resolución de un ejercicio de ilustración mediante dos vías: la primera (no recomendable y desarrollada a efectos puramente ilustrativos) sin utilizar el *truco kernel*; la segunda, como es recomendable, haciendo uso de dicho truco.

- El sexto tema es muy breve y está dedicado a la presentación del *software* que se empleará en los temas siguientes. Se presenta el entorno de programación R y los *packages* e1071 y kernlab. También se menciona la librería LIBSVM.

- El tema 7 se ocupa de la resolución con R, kernlab y e1071 de dos ejemplos de *support vector machines*: el primero de ellos es el ejemplo de ilustración presentado en la exposición teórica; el segundo es un análisis realizado a partir de una base que recoge datos sobre variables de competitividad en un conjunto de regiones europeas. En este tema se propone una representación gráfica alternativa a la de los *packages* e1071 y kernlab (ante los problemas detectados en ellos) y se proporciona (en un anexo) el código de R para obtener dicha representación gráfica.

- El octavo tema se dedica a resolver con R y kernlab un par de ejemplos de *kernel PCA*: un ejemplo ficticio creado *ad hoc* y un ejemplo sobre la base de datos de regiones europeas a la que se ha aludido. Al igual que en el tema anterior, se proponen representaciones gráficas alternativas a las proporcionadas por kernlab y se facilita el código de R para obtenerlas.

- A continuación, se recoge un comentario final.

- Seguidamente, se presenta la bibliografía consultada así como un conjunto de recursos *online* que han resultado útiles de cara a la redacción del documento.

- Por último, en sendos anexos, se presentan algunos elementos matemáticos esenciales en la definición y caracterización de las funciones *kernel* así como el código de R que se ha empleado para resolver los diferentes ejemplos.

Una última nota: en general, las traducciones al castellano de los términos originales (en inglés) empleados en el ámbito de las funciones *kernel* son muy variadas y, en ocasiones, inducen a la confusión. Por este motivo se ha optado, a lo largo de todo el documento, por mantener la terminología original inglesa, que será presentada con una tipografía *itálica*. Esto no es óbice para que, cada vez que aparezca en el documento un nuevo concepto, se proponga la traducción al castellano que consideremos más adecuada.

1. Introducción

Definición de aprendizaje estadístico

La búsqueda y detección[2] de regularidades, relaciones, pautas o patrones (*patterns*) en conjuntos de datos es el objetivo de la disciplina conocida como *aprendizaje estadístico*.

El aprendizaje estadístico surge de manera natural tan pronto como se dispone de un conjunto de datos sobre cierto fenómeno de interés. Es una herramienta de gran utilidad para tratar de determinar cuáles son las reglas de funcionamiento del mundo que nos rodea. Por ese motivo, el aprendizaje estadístico es de aplicación en muchos ámbitos de la actividad humana y, muy en particular, en las ciencias sociales y experimentales.

Aprendizaje supervisado frente a no supervisado

Dentro del aprendizaje estadístico se distingue tradicionalmente entre aprendizaje supervisado y aprendizaje no supervisado:

- En el aprendizaje supervisado, la tarea consiste en establecer un *mecanismo de clasificación* (en el caso de que la característica que se desee predecir sea cualitativa) o de *regresión* (en el caso de que la característica a predecir sea cuantitativa). Con ese fin, se dispone de una *muestra de entrenamiento etiquetada*, es decir, de un conjunto de observaciones dotadas cada una de ellas de un valor alusivo a la clase a la que pertenece (en el caso de la clasificación) o de un valor numérico correspondiente a la variable que se desea predecir (en el caso de la regresión). Una vez establecido el mecanismo predictor y evaluada su calidad (mediante, por ejemplo, el empleo de una *muestra test*) éste podrá aplicarse a otras *observaciones no etiquetadas* con la intención de predecir la clase a la que pertenecen o el valor cuantitativo de la variable a explicar. Son ejemplos de tareas de aprendizaje supervisado el análisis discriminante y la regresión lineal múltiple entre otros.

- En el aprendizaje no supervisado, la característica principal es la falta de etiqueta en los datos de que se dispone. No existe, en el conjunto de datos, una estructura previamente establecida de manera firme (de ahí la imposibilidad de asignar una etiqueta a cada una de las observaciones). Precisamente, el objetivo de las tareas de aprendizaje no supervisado es, en muchas ocasiones, tratar de detectar y hacer aflorar la existencia de cierta estructura en los datos, por ejemplo, entre las variables (como persigue el *análisis de componentes principales*) o entre los individuos (como es el caso del *análisis clúster* o *clasificación automática*).

2 En puridad, debería reservarse la expresión “detección de patrones” para las tareas de tipo no supervisado, en las que no existe una estructura previa en los datos, y emplear “identificación de patrones” para describir el objetivo de las tareas de tipo supervisado, en las que la meta es hacer manifiesta una estructura previamente existente. No obstante, por brevedad, a lo largo del documento emplearemos indistintamente una y otra expresión en la esperanza de que, en cada caso, el contexto aclare el significado.

Datos

Los datos son la materia prima del aprendizaje estadístico, pero no cuando se presentan de forma aislada. Los datos adquieren su valor cuando, siendo comparables, se aglutinan para conformar bases de datos. Es en las bases de datos donde los procedimientos de aprendizaje estadístico son capaces de detectar regularidades o pautas.

Conviene advertir desde el principio que la acepción de dato que se emplea en el ámbito del aprendizaje estadístico es muy comprehensiva, en el sentido de que va más allá de un significado puramente cuantitativo o vectorial. Concretamente, en el ámbito del aprendizaje estadístico se entiende por *dato* todo resultado obtenido a partir de cualquier método de observación, medición o registro que trate de determinar en qué cuantía (si se trata de una característica cuantitativa) o en qué modalidad (si se trata de una característica cualitativa o categórica) se da cierta propiedad (*variable*) en determinado objeto (al que se suele llamar *individuo*). En este sentido, tanto es un dato el valor de la cotización de determinada opción de compra en el mercado bursátil de Tokyo en una fecha determinada como los cien primeros términos del segundo capítulo del libro de cierto autor. En el ámbito del aprendizaje estadístico existen procedimientos (algoritmos) que permiten trabajar con datos de naturaleza tan diversa. Esta acepción tan amplia del concepto de "dato" en el ámbito del aprendizaje estadístico ha sido posible, como se pondrá de manifiesto, gracias al vertiginoso desarrollo de la disciplina en las últimas décadas.

Un cambio de paradigma

Vivimos, sin duda, en una sociedad interconectada y permanentemente *online*. Con cada una de nuestras acciones vamos dejando un reguero de datos que, gracias a esta interconexión, pueden ser registrados, almacenados, depurados y analizados (The Economist, 2010; Baker, 2008). Por ejemplo, la compra que hacemos en el supermercado, la página *web* que visitamos con nuestro ordenador o con nuestro teléfono e incluso el mero hecho de encender la luz en la habitación de los niños, son ejemplos de acciones cotidianas que generan conjuntos de datos que, si alguien deseara, podrían recogerse en una base de datos a la espera de ser analizados con el fin de detectar patrones de comportamiento que puedan ser explotados (por ejemplo, con fines comerciales). La realidad, sin embargo, es que gran parte de los datos que generamos nunca se registran ya que (quizás afortunadamente) la capacidad de almacenamiento (y no digamos de análisis) de datos es irrisoria si la comparamos con nuestra capacidad de generarlos.

Aun así, esta ubicuidad de los datos ha llevado a algunos autores (Anderson, 2008; Hey *et al.*, 2009) a advertir del advenimiento de lo que han dado en llamar la "Era del Petabyte" (*The Petabyte Age*). Estos autores vaticinan que la omnipresencia de los datos va a ser uno de los factores clave en el "Fin de la Ciencia" (*The End of Science*), un cambio de paradigma en el modo de entender la ciencia.

En efecto, la forma de entender la ciencia desde hace siglos ha seguido un enfoque basado en la teoría (*theory-driven approach*) que, en opinión de estos autores, estaría siendo paulatinamente reemplazado por un enfoque basado en los datos (*data-driven approach*).

Según el paradigma del *theory-driven approach,* la ciencia debe avanzar impulsada de modo exclusivo por la formulación de modelos basados únicamente en consideraciones de tipo teórico. Sería inapropiado afirmar que los datos no tienen ninguna función en este enfoque basado en la teoría, aunque sí se puede concluir que el papel que se les reserva es secundario. En efecto, los

defensores de este enfoque basado en la teoría consideran que, una vez formulado un modelo teórico, éste debe ser sometido a un contraste empírico. Así, el rol atribuido a los datos (y, en consecuencia, a las técnicas de análisis) queda limitado a servir de "piedra de toque" de las teorías salidas del magín de los científicos. Este enfoque basado en la teoría es el predominante hasta la fecha en muchas disciplinas. En efecto, es muy habitual que las investigaciones procedan del siguiente modo:

- Se comienza por la formulación de una serie de *hipótesis*, que normalmente reflejan la existencia de un conjunto de relaciones o asociaciones entre determinadas características de los elementos de un conjunto. Estas hipótesis han sido formuladas a partir de la revisión del *corpus* teórico relevante en la disciplina.

- A continuación, para cada una de las hipótesis planteadas, se construye un *contraste de hipótesis estadístico* mediante el que se establece una competición entre la hipótesis original y otra hipótesis (a la que se suele denominar *nula*) que expresa la inexistencia de relación entre las características consideradas.

- Seguidamente se procede a la recolección de datos (habitualmente mediante un *muestreo*) y, a partir de ellos, al cálculo de un *estadístico* y a la evaluación de su *probabilidad crítica* (o probabilidad - bajo el supuesto de que se cumple la hipótesis nula- de obtener un valor más extremo que el obtenido para el estadístico en la muestra concreta).

- En función de cuál sea la probabilidad crítica y del *nivel de significación* elegido por el investigador se toma una decisión acerca de la veracidad o falsedad de la hipótesis nula (y, en consecuencia, de la hipótesis originalmente formulada).

Son muchos los detractores de esta forma de proceder. Por ejemplo, Ziliak y McCloskey (2008) o Schrodt (2010) consideran que este enfoque está llevando a decisiones equivocadas, que están suponiendo pérdidas económicas y sociales difícilmente reparables.

En otras palabras, el paradigma del *theory-driven approach*, propone el siguiente esquema:

- La ciencia debe avanzar a partir de hipótesis contrastables (modelos), que no son sino conjeturas salidas de la mente de los científicos.

- Los modelos deben enfrentarse con los datos y ser confirmados o refutados mediante experimentos.

- Mientras no sean refutados, los modelos constituyen la explicación "oficial" acerca de "cómo funciona el mundo", si bien se trata de una explicación provisional en el sentido de que siempre cabe que el modelo propuesto no pueda proporcionar una explicación satisfactoria a algún aspecto de la realidad. En ese momento, el modelo habrá sido refutado y será necesario construir uno nuevo que dé cabida a ese hecho empírico. De hecho, el procedimiento de refutación es la forma de avance de la ciencia en este enfoque basado en la teoría.

En este enfoque, los modelos son los protagonistas y los datos se limitan a un papel secundario. Un ejemplo puede resultar ilustrativo: es habitual instruir a los científicos en la distinción de *correlación* y *causalidad.* En concreto, se hace mucho hincapié en señalar que no es aceptable

concluir que X cause Y sólo porque exista una correlación entre X e Y. Sin embargo, una vez que existe un modelo teórico que afirma que X causa Y, una correlación entre X e Y es suficiente para afirmar que existe la relación de causalidad entre ellas. Como pone de manifiesto este ejemplo, en el enfoque basado en la teoría, los datos, sin el modelo, son puro ruido; sin embargo, acompañados de un modelo, los datos tienen fuerza probatoria. Este es el modo en que ha funcionado la ciencia durante largo tiempo.

En el nuevo paradigma del *data-driven approach* (surgido al calor de la abundancia de datos y del desarrollo de procedimientos para extraer patrones de ellos) el rol que juegan los datos en el progreso de la ciencia es mucho más protagonista. Los datos no se limitan a ser "piedra de toque" de las formulaciones teóricas sino que extienden su ámbito de actuación para colaborar en la propia formulación de las hipótesis. Algunos, incluso llegan al extremo de prescindir del modelo. Por ejemplo, Peter Norvig, director de investigación de Google, señalaba en marzo de 2008[3] parafraseando a G. Box: "Todos los modelos están equivocados y cada vez es más posible tener éxito sin contar con ellos"[4].

Ante volúmenes masivos de datos, el *theory-driven approach* resulta progresivamente más obsoleto. Las grandes bases de datos nos permiten decir: "quizás la correlación sea suficiente", "quizás no necesitemos modelos". Podemos analizar los datos sin ideas preconcebidas acerca de qué nos podemos encontrar en ellos. Quizás, si dejamos que los ordenadores "trituren" esta enorme cantidad de datos, es posible que encontremos patrones en ellos que ni siquiera hubiéramos sido capaces de imaginar.

El cuarto paradigma

En esta misma línea, el desaparecido Jim Gray (científico estadounidense galardonado en 1998 con el premio Turing) fue capaz de anticipar esta nueva situación a mediados de la década de los noventa. Según Hey *et al.* (2009), Gray acuñó el término *eScience* para referirse al nuevo paradigma imperante en la investigación científica: la investigación intensiva en datos (*data-intensive scientific research*). En opinión de este autor, la investigación científica ha atravesado cuatro estadios (paradigmas):

- El *paradigma empírico*. En los albores de la ciencia, ésta era puramente *empírica* y su objetivo no era otro que el de tratar de describir los fenómenos naturales. En particular, los científicos no pretendían generar modelos o leyes que explicaran comportamientos universales sino tan solo explicar aquello que veían.

- El *paradigma teórico*. La ciencia evolucionó y sus objetivos se tornaron más ambiciosos. La ciencia no se limitaba ya a tratar de explicar aquello que veía. Los científicos deseaban formular modelos de aplicación general: modelos universales. En otras palabras, a pesar de que la ciencia continuó siendo empírica, adquirió un carácter teórico. Las leyes de Kepler, las leyes de Newton o las ecuaciones de Maxwell son ejemplos apropiados de este estadio de la ciencia.

- El *paradigma de la simulación*. Los modelos teóricos universales que formulaban los

3 http://en.oreilly.com/et2008/public/schedule/detail/1778

4 "*All models are wrong and increasingly you can succeed without them*". La frase original, atribuida a George Box, es "*All models are wrong, but some are useful*".

científicos en su esfuerzo por explicar el funcionamiento de la realidad se fueron volviendo progresivamente más complejos, hasta el punto de que pronto resultó evidente que eran demasiado complicados como para poder ser resueltos de modo analítico (exacto). Los científicos recurrieron entonces a la *simulación*. La Física, la Biología o la Química (en el ámbito de las Ciencias Experimentales) o la Economía y la Psicología (entre las Ciencias Sociales) son ejemplos de disciplinas que han recurrido profusamente a la simulación (apoyada normalmente en el recurso intensivo a los ordenadores) con el fin de valorar el desempeño de sistemas excesivamente complejos como para ser evaluados mediante un enfoque analítico.

- El *paradigma intensivo en datos*. En la actualidad, el creciente recurso a la simulación ha generado grandes volúmenes de datos. A éstos se une el aluvión proveniente de las observaciones empíricas de las Ciencias Experimentales (que cuentan con instrumentos cada vez más precisos y potentes). Todos esos datos, captados o generados, se acumulan en enormes bases de datos a la espera de ser analizados mediante potentes programas de ordenador que persiguen la detección de patrones o pautas de regularidad. Según este nuevo paradigma, estas pautas permitirán a los científicos aventurar nuevas conjeturas acerca de cómo funciona el mundo. Esta forma de entender la ciencia recibe el nombre de *El Cuarto Paradigma*.

Tanto el enfoque del "*data-driven approach*" como la reflexión sobre el llamado "*Cuarto Paradigma*" son dos vías alternativas de describir una misma tendencia: el auge de los mecanismos automáticos para la detección de patrones en bases de datos como método para promover el avance de la ciencia[5].

Con el fin de profundizar en las ideas que se han apuntado debemos comenzar por definir con precisión qué es exactamente lo que queremos decir cuando hablamos de un *patrón* en un conjunto de datos.

Patrones en conjuntos de datos

Cualquier pauta, regularidad o relación existente en un conjunto de datos recibe el nombre de *patrón*.

Pensemos, por ejemplo, en una base de datos que recoja para cada una de las compras realizadas en una cadena de supermercados a lo largo del año 2009 el número de unidades de cada uno de los productos del catálogo que se han incluido en la citada compra. Podemos concebir esta base de datos como una inmensa matriz con tantas filas (registros) como compras se hayan producido en la cadena de supermercados durante 2009 y tantas columnas (campos) como artículos consten en el catálogo de la cadena. En cada una de las casillas se registraría el número de unidades del producto correspondiente a la columna que se han adquirido en la compra correspondiente a la fila. Imaginemos que, una vez analizada esta base de datos, se detecta la existencia de una relación (correlación) positiva entre el número de pañales y el número de productos precocinados adquiridos. Esta relación constituiría un patrón[6].

---

5 En esta misma línea, resulta sumamente esclarecedor el trabajo de L. Breiman de título *Statistical Modeling: The Two Cultures* publicado en *Statistical Science* (2001), vol 16, nº 3, págs. 199 a 231.

6 Los responsables de la cadena de supermercados podrían aprovechar el patrón detectado para ofrecer un *pack* de pañales y alimentos precocinados a los estresados padres de niños lactantes que no tienen tiempo o ánimos para cocinar.

Naturalmente, para que sea posible detectar patrones, es necesario que los datos no estén aislados. Dicho de otro modo, es imposible encontrar patrones en un único dato. Sólo a partir de conjuntos de datos comparables es posible detectar regularidades, pautas o relaciones que permitan formular conjeturas acerca de determinados fenómenos de interés.

Decir que en un conjunto de datos existe un patrón o pauta es equivalente a afirmar que existe *redundancia* en ellos. Es precisamente esta redundancia la que hace posible reconstruir partes del conjunto de datos a partir de otras partes (la redundancia de la base de datos da lugar a que ésta se pueda comprimir o resumir) o incluso predecir o estimar los valores de algunos datos en función de los valores de otros datos para objetos que no han sido estudiados (la redundancia proporciona predictibilidad).

Podemos, por tanto, definir un patrón como cualquier regularidad, relación o pauta entre los elementos de un conjunto de datos que genere redundancia en ellos.

Aprendizaje estadístico automático

Como ya se ha mencionado, los datos adquieren su valor cuando se ponen en relación con otros datos relativos a las mismas características (variables) obtenidos como resultado de estudiar objetos (individuos) comparables entre sí. Naturalmente, cuanto mayor es el tamaño del conjunto de datos, mayor es su riqueza[7]. Sin embargo, el valor de los datos no es la única de sus propiedades que se incrementa con el tamaño: la dificultad para detectar patrones también crece con el tamaño de la base de datos. Ante esta realidad, resulta comprensible que los primeros procedimientos de detección de patrones se centraran en bases de tamaño reducido así como en la búsqueda de patrones sencillos.

De entre todos los patrones que cabe encontrar en un conjunto de datos, los más simples son los de tipo lineal. En efecto, la tarea de detección de patrones lineales en conjuntos de datos se apoya en las técnicas del álgebra lineal, disciplina que proporciona soluciones exactas (analíticas) a estos problemas. Estas técnicas funcionan de manera muy eficiente cuando se enfrentan a bases de datos de tamaño reducido. Sin embargo, desde los primeros momentos se puso de manifiesto su inaplicabilidad a bases de datos de miles de registros (filas) y cientos de campos (columnas), como resultado del hecho de que la *complejidad temporal* de muchos de los procedimientos básicos del álgebra lineal es cúbica (lo que significa que el tiempo necesario para resolver un problema de tamaño doble se multiplica aproximadamente por ocho). La implicación de este hecho es evidente: para bases de datos de tamaño moderado el enfoque analítico es inaceptable ya que lleva a tiempos de proceso muy elevados.

En consecuencia, se manifestó la necesidad de desarrollar *técnicas aproximadas* para la detección de patrones lineales. Estas técnicas aproximadas están basadas en procedimientos de optimización convexa (*convex optimization*), muy bien estudiados y con propiedades muy deseables. Por ejemplo, para este tipo de patrones, no cabe la existencia de óptimos locales que pueden llevar a adoptar como solución óptima un conjunto de valores muy alejado del óptimo global.

En cualquier caso, tanto si se detectan a partir de procedimientos analíticos (exactos) o a través de métodos aproximados, los patrones de tipo lineal resultan muy intuitivos ya que, a partir de ellos, es

7 Conviene comentar que esta afirmación no está carente de límites: como es bien conocido, no siempre una muestra de mayor tamaño es mejor. Es necesario prestar atención a su representatividad y a la calidad en el proceso de recogida de los datos.

posible formular modelos de la realidad basados en relaciones de proporcionalidad entre diferentes conceptos. Dado que la noción de proporcionalidad se encuentra profundamente interiorizada, los patrones de tipo lineal resultan de muy fácil interpretación.

Lamentablemente, la sencillez de un modelo va de la mano de la rigidez. En otras palabras, pretender que sólo cabe encontrar patrones de tipo lineal en las bases de datos analizadas refleja una actitud análoga a la de quien habiendo perdido sus llaves se limita a buscarlas cerca de una farola porque allí hay luz, aun a sabiendas de que lo más probable es que las haya perdido muy lejos de dicho lugar. En otras palabras, dada la complejidad de las relaciones existentes en el mundo real, la limitación del espacio de búsqueda de patrones a aquellos de tipo lineal se fue haciendo progresivamente menos aceptable entre los investigadores. Surgieron así voces que reclamaban el desarrollo de procedimientos para la detección de patrones de naturaleza no lineal.

Desgraciadamente, a diferencia de lo que ocurre con los patrones lineales, no existen, en general, soluciones analíticas (exactas) para los problemas de detección de patrones no lineales y, por tanto, es necesario recurrir a procedimientos aproximados. Además, estos procedimientos aproximados no gozan de las ventajas de los métodos de detección de patrones lineales. Por ejemplo, es muy habitual que se produzcan óptimos locales muy alejados de los óptimos globales, con los evidentes problemas que de este hecho pueden derivarse.

En todo caso, bien sea por el incremento en el tamaño de las bases de datos, bien sea por la extensión de los posibles patrones a los de tipo no lineal, el enfoque analítico (exacto) fue abandonado y los investigadores encargados de detectar patrones en bases de datos abrazaron los métodos aproximados, al calor del desarrollo de la herramienta que ha cambiado nuestra vida en las últimas décadas: el ordenador. La detección de patrones pasó a ser un procedimiento automático que, *grosso modo*, consistía en programar un ordenador con cierto algoritmo, proporcionarle un conjunto de datos y dejar que trabajara. El incremento en la capacidad de procesamiento de datos de los ordenadores (que alcanza en algunos casos el *petaflops*[8]) ha permitido que las bases de datos analizadas sean cada vez de mayor tamaño y los patrones buscados de una mayor complejidad.

En resumen, el aprendizaje estadístico se ha convertido, por obra y gracia del desarrollo y ubicuidad de los ordenadores, en un procedimiento de detección automática de patrones. La detección de patrones en bases de datos se lleva a cabo mediante procedimientos automáticos, establecidos detalladamente en algoritmos y traducidos en programas que son ejecutados de forma eficiente por los ordenadores.

Características deseables en un algoritmo de detección de patrones

Una vez establecido que la detección de patrones se realiza a través del diseño de algoritmos que son finalmente ejecutados por ordenadores, conviene determinar cuáles son las características deseables en un algoritmo diseñado con este propósito. Entre las más importantes cabe citar:

- *Eficiencia computacional*: La detección de patrones en bases de datos es una tarea eminentemente práctica. En consecuencia, los algoritmos que se diseñen para este fin deben ser capaces de dar respuesta a los problemas que se planteen, más allá de cuál sea el tamaño particular de la base de datos sobre la que se vaya a aplicar el procedimiento. Dicho de otro

8 Un *petaflops* (PFLOPS) equivale a 10^{15} *operaciones de punto flotante por segundo*. El término FLOP es un acrónimo inglés a partir de la expresión *FLoating point Operations per Second*.

modo, un algoritmo que funcione bien (es decir, que sea capaz de detectar patrones en un tiempo razonable) en bases de datos de reducido tamaño (con unas pocas decenas de filas y unas pocas columnas) pero fracase en bases de datos reales (por presentar tiempos de ejecución muy elevados), no resultaría aceptable. De forma más precisa, debe exigirse que un algoritmo de detección de patrones sea *eficiente desde el punto de vista computacional*. Son muchas las formas de medir la eficiencia de un algoritmo pero, quizás, la más extendida sea la evaluación de su *complejidad temporal.* La complejidad temporal de un algoritmo es una expresión del tiempo de ejecución (o del número de operaciones) que precisa dicho algoritmo en función del tamaño del problema (medido, por ejemplo, a partir del número de filas o columnas de la base de datos analizada). Para expresar la complejidad temporal de un algoritmo suele recurrirse a la llamada *notación de Landau* (o *notación de la O mayúscula*). Se dice que un problema es *tratable* cuando el algoritmo más eficiente para resolverlo tiene complejidad temporal polinómica o inferior, es decir, cuando la función que pone en relación el tamaño del problema n con el tiempo (o número de operaciones) que requiere el algoritmo para resolverlo pertenece a $O(n^k)$ con $k \in \mathbb{N}$. Este requisito es, en realidad, bastante laxo ya que complejidades polinómicas pueden suponer (si el grado del polinomio k es alto) tiempos de ejecución que hoy en día (a pesar del rápido incremento en la velocidad de los procesadores) resultan inaceptables. Lamentablemente, muchos de los algoritmos que se diseñan para la detección de patrones presentan complejidades temporales superiores a la polinómica y son, por tanto, intratables desde el punto de vista de la teoría de la complejidad algorítmica.

- *Robustez*: Cualquier base de datos, y en mayor medida si es de gran tamaño, es susceptible de contener valores erróneos. En ocasiones, estos valores erróneos serán detectados (y convenientemente corregidos) por los procedimientos de detección de *outliers*, pero en ocasiones, aun a pesar de los mayores y mejores esfuerzos por parte de los analistas, los datos erróneos escaparán esta criba preliminar y comprometerán la calidad de los análisis que se realicen tomando como materia prima la base de datos en la que se encuentran. Las fuentes de las que pueden provenir estos datos erróneos (a las que se suele denominar fuentes de *errores sistemáticos*) son muchas y de muy diversa naturaleza. Entre otras cabe citar los problemas en los mecanismos de obtención de los datos (por ejemplo, errores en los instrumentos de medida, que pueden no estar correctamente calibrados), errores en la transmisión de los datos de unos soportes a otros (por ejemplo, al teclear manualmente los datos de un cuestionario en papel a una base de datos electrónica) o también errores en la preparación de los datos con carácter previo a la realización de algún análisis. Sin ninguna duda, el mejor antídoto contra este tipo de errores es un cuidado exquisito por parte del encargado de alimentar o manipular la base de datos en cada uno de los pasos descritos, pero, a pesar de los mejores esfuerzos por su parte, es imposible evitarlos por completo. De este modo, resulta evidente que una característica muy deseable en los algoritmos automáticos de detección de patrones es que éstos sean relativamente insensibles a la presencia de una cierta proporción de datos erróneos. A esta propiedad se le llama *robustez*. Así, se dice que un algoritmo de detección de patrones es robusto cuando los patrones detectados en una base de datos no se ven alterados (o al menos no de forma notable) cuando en ésta existe una pequeña proporción de datos erróneos. Lamentablemente, esta propiedad de robustez entra en conflicto con otra propiedad deseable: la *sensibilidad*, por la que un algoritmo automático debería ser capaz de detectar patrones diferentes en bases de datos diferentes.

- *Estabilidad*: Si entendemos una base de datos como una manifestación concreta (particular) de un determinado mecanismo generador de datos, es cuando el concepto de *estabilidad de*

un algoritmo cobra todo su significado. Dicho de forma breve: los algoritmos de detección de patrones deberían detectar pautas propias de los mecanismos generadores de datos y no las pautas particulares de la base de datos concreta que están analizando. Con mayor precisión: diremos que un algoritmo de detección de patrones es *estable* si obtenidas dos bases de datos alternativas mediante un mismo mecanismo generador (llamémoslas B1 y B2), el algoritmo detecta los mismos patrones en B1 y en B2. Así, un algoritmo estable es aquél que detecta los patrones que corresponden al mecanismo generador de los datos y que obvia las particularidades específicas de la base de datos concreta que se está sometiendo a análisis.

A este respecto resulta muy ilustrativo el comentario de Paulos (1990) en su obra *Innumeracy* (traducida al español como *El Hombre Anumérico*), quien cita a Plutarco cuando afirma "*It is not great wonder if, in the long process of time, while fortune takes her course hither and thither, numerous coincidences should spontaneously occur*". Se trata de una advertencia sobre el riesgo de lo que dos mil años más tarde se ha dado en llamar *overfitting* (y que ha sido traducido al castellano como *sobreajuste*): el riesgo de que un algoritmo de detección de patrones se ajuste en exceso a la realidad concreta de una base de datos identificando, de este modo, pautas que no corresponden al mecanismo generador de los datos sino que son particulares de la base de datos y, en consecuencia, no generalizables.

El mismo Paulos pone de manifiesto (en un ejemplo que más tarde recogen Shawe-Taylor y Cristianini, 2004) cómo la única forma de protegerse del riesgo de *overfitting* es mediante la limitación de los patrones que cabe esperar encontrar en la base de datos. Si el analista de los datos está dispuesto a encontrar cualquier patrón (y diseña el algoritmo con este criterio) es seguro que encontrará algún patrón que le resultará sorprendente: "*The paradoxical conclusion is that it would be very unlikely for unlikely events not to occur*". El ejemplo al que recurren los mencionados autores para ilustrar este fenómeno es la archiconocida *Paradoja del Cumpleaños*, que pone de manifiesto cómo son suficientes 23 personas reunidas en una sala para que al menos dos de ellas compartan su fecha de cumpleaños (una fecha cualquiera del año) mientras que son necesarias 253 personas para que alguna cumpla años en una fecha concreta establecida a priori. La clave de esta diferencia reside en los diferentes espacios de patrones que se están tomando en consideración (*un día concreto* es mucho más restrictivo que *cualquier día del año*).

Un ejemplo adicional puede ayudar a ilustrar la idea de *overfitting*: imaginemos que contamos con $n \in \mathbb{N}$ parejas de observaciones $(x_i, y_i) \in \mathbb{R}^2$ tales que $x_i \neq x_j \ \forall i \neq j$ y supongamos que deseamos establecer un patrón que permita predecir el valor de y_i en función del valor de x_i. Pues bien, si el espacio de patrones es el conjunto de polinomios de grado $k \in \mathbb{N}$ y estamos dispuestos a incrementar cuanto sea necesario el valor de k, será posible encontrar un patrón (en concreto, dicho patrón será un polinomio de grado $n-1$) que prediga de forma perfecta cada uno de los valores y_i en función del correspondiente x_i [9]. Ahora bien, si mediante el mismo mecanismo de generación de datos obtuviéramos otro conjunto de observaciones $(x_i, y_i) \in \mathbb{R}^2$ y aplicáramos el patrón aprendido en la base original para tratar de efectuar predicciones en la nueva base, los

9 La demostración de este hecho se basa en que, tal como se ha planteado el problema, la determinación del polinomio de grado $n-1$ consiste en la resolución de un sistema de n ecuaciones con n incógnitas en el que la matriz de coeficientes es un *matriz de Vandermonde* con determinante no nulo. En consecuencia, el sistema de ecuaciones es compatible determinado. El polinomio así establecido recibe el nombre de *polinomio de interpolación*.

resultados serían mucho menos espectaculares. El conjunto de patrones susceptibles de ser localizados era excesivamente grande y, por tanto, ha ocurrido el fenómeno de *overfitting*.

En aprendizaje estadístico, la llamada *complejidad de Rademacher* mide la *riqueza* de una familia de funciones (posibles patrones), entendiendo esta riqueza como la capacidad de la familia de funciones a adaptarse (y detectar) *ruido aleatorio*. Cuanto mayor sea la complejidad de Rademacher de una familia de funciones, mayor es el riesgo de *overfitting* que se deriva de su utilización.

Como se ha mencionado ya, la única vía para evitar el problema del *overfitting* es mediante la limitación o restricción del espacio de búsqueda de posibles patrones. Con este fin, es imprescindible contar con *conocimiento experto* en la disciplina a la que se refiere la base de datos analizada. Sólo un experto en el ámbito de estudio (apoyado por el diseñador del algoritmo) puede decidir de manera informada y cabal cuál es la naturaleza de los patrones que cabe detectar en su área de conocimiento así como señalar qué patrones no son razonables. De esta manera, el diseñador del algoritmo estará en las condiciones óptimas para poder adaptar el algoritmo al ámbito al que se refiere la base de datos analizada. En relación con esta reflexión acerca de la incorporación de conocimiento experto a los algoritmos automáticos de detección de patrones surge una nueva propiedad deseable en ellos: la *modularidad* o posibilidad de parametrizar un algoritmo de modo que, a partir de un núcleo central común, sea posible adaptarlo a problemas de distinta naturaleza.

La solución kernel

De todo lo anterior, puede concluirse que los diseñadores de algoritmos de detección de patrones deberían decantarse por procedimientos que, entre otras propiedades:

- Disfruten de las ventajas computacionales propias de los algoritmos lineales. En particular, deberían tratar de que los algoritmos no descansen en procedimientos de optimización que puedan verse perjudicados por la presencia de óptimos locales alejados de los óptimos globales (tal como ocurre, en muchos casos, en los algoritmos de detección de patrones no lineales).

- Sean robustos, en el sentido de que no se vean influidos en exceso por la presencia de una pequeña proporción de datos erróneos. De manera ideal, sería preferible que esta robustez no afectara a la sensibilidad de los algoritmos.

- Gocen de la flexibilidad que proporcionan los algoritmos de detección de patrones no lineales, que permiten la determinación de pautas y relaciones más complejas que las proporcionadas por los rígidos modelos lineales.

- Faciliten la incorporación sencilla de conocimiento experto como única vía para limitar el espacio de patrones posibles y evitar, de ese modo, el fenómeno de *overfitting*.

En definitiva, si deseáramos adoptar un nuevo enfoque en la elaboración de algoritmos de detección de patrones y quisiéramos que este nuevo enfoque estuviera dotado de todas estas características que hemos calificado de deseables, deberíamos mirar tanto a la familia de los algoritmos lineales como a la de los no lineales para tomar lo mejor de cada uno de estos mundos. En efecto, esto es lo que consigue el enfoque conocido como *aprendizaje estadístico con funciones kernel*.

En esencia, un algoritmo de detección de patrones basado en funciones *kernel* consiste en:

- Un conjunto de elementos en el que se desea explorar la existencia de patrones o pautas. Llamaremos a este conjunto *Espacio de Entrada* (*Input Space*) y lo denotaremos por X. No es necesario que este conjunto esté dotado de una estructura algebraica particular; puede tanto ser un espacio vectorial (en el que los elementos de la base de datos aparecen representados en función de los valores que toman en un conjunto de variables de naturaleza cuantitativa) como un conjunto sin dicha estructura. Como veremos, la clave del enfoque *kernel* consiste en que sea posible definir una función que a cada pareja de elementos de este espacio X le haga corresponder un valor real y para definir esa función sobre el producto cartesiano $X \times X$ no es necesario que X sea un espacio vectorial. La gran variedad de espacios de entrada sobre los que se puede aplicar la metodología *kernel* es una de sus propiedades más interesantes: los métodos *kernel* gozan de una gran *versatilidad*.

- Un conjunto de patrones al que llamaremos *Espacio de Características* (*Feature Space*), que debe tener una estructura algebraica de *Espacio de Hilbert* y que denotaremos por F. En particular, el espacio F debe estar dotado de un *producto escalar*:

$$<,> : F \times F \rightarrow \mathbb{R}$$

- Una función $\phi : X \rightarrow F$ que incrusta (*embeds*) cada elemento del *input space* X en el *feature space* F. Esta función ϕ recibe el nombre de *embedding*.

- Un algoritmo de detección de patrones lineales en el *feature space* F que ha sido *kernelizado*, es decir, ha sido rediseñado de manera tal que para alcanzar su objetivo de detección de patrones en F el algoritmo no necesita disponer de los valores concretos de $\phi(x) \ \forall x \in X$ sino tan solo de los productos escalares entre las imágenes de los elementos de X, es decir, $<\phi(x), \phi(z)> \ \forall x, z \in X$, siendo $<,> : F \times F \rightarrow \mathbb{R}$ el producto escalar definido en F al que se ha aludido previamente.

- Una función real definida sobre el producto cartesiano del *input space* con él mismo $k : X \times X \rightarrow \mathbb{R}$, llamada función *kernel*, que a cada pareja de elementos del *input space* X le hace corresponder el producto escalar en F de sus respectivas imágenes por la función ϕ, es decir, tal que:

$$k(x, z) = <\phi(x), \phi(z)> \quad \forall x, z \in X$$

  Podemos entender la función *kernel* como una "pseudocomposición" del *embedding* ϕ y del producto escalar $<,> : F \times F \rightarrow \mathbb{R}$ que para cada pareja (x, z) de elementos del conjunto X proporciona directamente $<\phi(x), \phi(z)>$ sin necesidad de transitar por el *embedding* ϕ ni por el producto escalar $<,>$. Este atajo de $X \times X$ a $\mathbb{R}$ junto con la *kernelización* aludida en el punto anterior conforman el llamado *truco kernel* (*kernel trick*), que es la base del método.

Este sencillo esquema proporciona una inteligente solución al dilema entre los algoritmos de detección de patrones lineales o no lineales, reuniendo en un nuevo enfoque las mejores características de cada uno de los dos mundos:

- En efecto, los procedimientos de detección de patrones basados en funciones *kernel* son, en esencia, métodos de detección de patrones lineales en el espacio F y gozan, en

consecuencia, de todas las ventajas que para estos algoritmos han sido comentadas (sencillez, estabilidad en la solución, eficiencia computacional,...).

- Al mismo tiempo, gozan de la flexibilidad de los algoritmos de detección de patrones no lineales gracias al *cambio en la representación de los datos*. Como resulta evidente, la elección de la función de *embedding* (función ϕ) hace que los patrones lineales detectados en el *feature space* correspondan a patrones potencialmente no lineales en el *input space*. Así, mediante la detección de patrones lineales (mediante sencillos procesos de optimización convexa con garantías de localización de óptimos globales en tiempos razonables) en el espacio F, un algoritmo de detección de patrones basado en funciones *kernel* está, en realidad, detectando patrones no lineales en el espacio original X.

- La elección de la función de *embedding* ϕ permite la incorporación de conocimiento experto y, por tanto, aporta modularidad y adaptabilidad al algoritmo. En efecto, la función de *embedding* elegida determina el conjunto de patrones que cabe esperar en la base de datos analizada. No obstante, más adelante veremos que, a pesar de que esta afirmación es cierta, este papel de ser la vía de incorporación de conocimiento experto a los algoritmos queda finalmente asignado no al *embedding* sino a la función *kernel* propiamente dicha.

- Como veremos, la dimensión del *feature space* puede ser muy elevada (incluso infinita). Esto suscita, a primera vista, un problema de eficiencia computacional, ya que tanto el cálculo de las imágenes por ϕ de los elementos de X como la búsqueda de patrones (aunque sean sencillos patrones lineales) en un espacio de elevada dimensión pueden resultar muy consumidores de tiempo. Los procedimientos de detección de patrones basados en funciones *kernel* solventan este problema recurriendo al previamente aludido truco *kernel* (*kernel trick*), que se apoya en la *kernelización* de los algoritmos de búsqueda de patrones lineales y en la existencia de la función *kernel* para evitar el cálculo de las imágenes de los elementos de X así como para simplificar la detección de patrones en F.

    - En efecto, la *kernelización* de un algoritmo de detección de patrones lineales consiste en su reformulación, de manera que la determinación de una pauta o regularidad lineal en los datos pueda llevarse a cabo a partir, exclusivamente, de la información recogida en los productos escalares calculados para todas las parejas de elementos del espacio. El conjunto de dichos productos escalares recoge, en esencia, la información existente en el conjunto de datos relativa a las normas de los elementos del espacio así como a los ángulos que existen entre ellos. Aunque resulta evidente que prescindir de las coordenadas reales de los elementos en el espacio y limitarse a la información recogida en el conjunto de productos escalares supone una pérdida de información (por ejemplo, se pierde la información relativa a la orientación del conjunto de datos en el espacio o la relativa a la alineación de los elementos con las variables originales), en muchas ocasiones esta pérdida no es relevante para alcanzar el objetivo de detección de patrones lineales. En otras palabras, el conjunto de productos escalares entre los elementos del espacio es, muy habitualmente, un *estadístico suficiente* para la determinación del patrón lineal, y esta suficiencia se pone de manifiesto cuando se logra la *kernelización* del algoritmo. Afortunadamente, muchos algoritmos de detección de patrones lineales pueden ser *kernelizados*.

    - Por otra parte, la función *kernel* $k : X \times X \rightarrow \mathbb{R}$ hace corresponder a cada pareja de elementos del *input space* el producto escalar en F de las imágenes por ϕ de los

elementos de la pareja, es decir, $k(x,z)=<\phi(x),\phi(z)> \ \forall x,z\in X$.

- Así, la función *kernel*, que permite el cálculo directo de los productos escalares de las imágenes sin necesidad de calcular las imágenes propiamente dichas, junto con la *kernelización* del algoritmo de búsqueda de patrones lineales permiten prescindir por completo del *embedding* ϕ que, en muchas ocasiones, no es ni siquiera conocido.

- En consecuencia, la incorporación de conocimiento experto para la restricción del espacio de patrones a aquellos que es razonable encontrar en el conjunto de datos dada la disciplina a la que se refieren, se traslada de la función de *embedding* a la función *kernel*. En otras palabras, la elección de la función *kernel* es la que va a determinar qué patrones cabe esperar en el *input space*.

- Además, como se detallará más adelante, el *Teorema de Representación* garantiza que dada una función *kernel* $k: X\times X\rightarrow \mathbb{R}$ con la propiedad de ser *finitamente semidefinida positiva,* existe un espacio de Hilbert F y un *embedding* ϕ tales que:

$$k(x,z)=<\phi(x),\phi(z)> \ \forall x,z\in X$$

donde el producto escalar es el definido en F . En el espacio de Hilbert F se cumple la propiedad de que $<f,k(x,.)>=f(x) \ \forall f\in F$, lo que convierte a F en un *Reproducing Kernel Hilbert Space* (o RKHS), concepto definido por Aronszajn y Bergman en 1950 y aplicado al ámbito del aprendizaje estadístico en la década de los noventa . Este teorema, o alternativamente, el *Teorema de Mercer*, permiten prescindir tanto de F como de ϕ y centrar la atención en la función *kernel*, $k: X\times X\rightarrow \mathbb{R}$.

Evolución histórica del aprendizaje estadístico basado en funciones *kernel*

El desarrollo del aprendizaje estadístico basado en funciones *kernel* es, hasta 2000, paralelo al surgimiento y desarrollo de las *support vector machines*[10]. Este desarrollo de las *support vector machines* se produce en dos vías, que finalmente confluyen.

- Por una parte, se producen aportaciones en el campo de los algoritmos de detección de patrones lineales y, más en concreto, en el ámbito de la clasificación dentro del aprendizaje de tipo supervisado:

  - Fisher (1936) propuso el primer algoritmo para la detección de patrones.

  - El primer algoritmo de detección de patrones, de naturaleza aún lineal, que contiene la esencia de los clasificadores de margen máximo se debe a Vapnik y Lerner (1963): es el llamado *Generalized Portrait Method*. Este método fue mejorado más adelante por Vapnik y Chervonenkis (1964).

10 Una *support vector machine* (o máquina de vector soporte) es, en esencia, un conjunto de hiperplanos afines en un espacio de dimensión muy elevada (incluso infinita). Dicho conjunto de hiperplanos afines permite clasificar un conjunto de elementos que han sido incrustados (*embedded*) en el espacio aludido. Se puede concebir una *support vector machine* como la adaptación modular de un *clasificador de margen máximo* y de una *función kernel*.

- Cover (1965), Mangasarian (1965) y Duda y Hart (1973) estudian los hiperplanos de margen máximo en el *input space*.

- Smith (1968) resuelve el problema de la no separabilidad de las clases mediante la introducción de variables de holgura. Su aportación fue mejorada por Bennett y Mangasarian (1992).

- Por otra parte, se producen avances en la definición del concepto de *función kernel* y surgen voces para su aplicación en el campo del aprendizaje estadístico.

  - Aronszajn (1950), desarrollando las aportaciones de Mercer (1909), introdujo el concepto de *Reproducing Kernel Hilbert Space.*

  - Aizerman, Braverman y Rozonoer (1964) presentan la interpretación geométrica de los *kernels* como productos internos en el *feature space*.

  - Poggio y Girosi (1990), entre otros, proponen el uso de funciones *kernel* en el ámbito del aprendizaje estadístico.

- Finalmente, estas dos líneas de avance confluyen y dan lugar a la aparición de las *support vector machines*:

  - Puede decirse que la *teoría del aprendizaje estadístico* tiene su nacimiento con el trabajo de Vapnik y Chervonenkis (1974, publicado en ruso y traducido posteriormente al alemán por los mismos autores).

  - El desarrollo que Vapnik (1979) imprimió a esta teoría, anuncia el nacimiento de las *support vector machines.* Esta obra es traducida posteriormente a inglés (Vapnik, 1982).

  - La primera formulación de las *support vector machines* en su versión actual llegó de la mano de Boser, Guyon y Vapnik (1992).

  - Poco después, Cortes y Vapnik (1995) extendieron las *support vector machines* al caso del clasificador de margen máximo *soft*.

  - El propio Vapnik (1995) extendió la aplicación de las *support vector machines* al caso de la regresión.

  - Finalmente, los trabajos de Shawe-Taylor *et al.* (1998) y Shawe-Taylor y Cristianini (2000) proporcionaron cotas estadísticas a la generalizabilidad de las *support vector machines* en los casos *hard* y *soft* respectivamente.

Tras la llegada de las funciones *kernel* al ámbito del aprendizaje estadístico a través de las *support vector machines*, los últimos años han sido testigo de la proliferación de artículos científicos centrados en la *kernelización* de las técnicas lineales más conocidas: análisis discriminante lineal de Fisher (Baudat y Anouar, 2000), análisis de componentes principales (Schölkopf, Smola y Müller, 1998), análisis de correlación canónica (Van Gestel, Suykens, De Brabanter, De Moor y Vandewalle, 2001), clasificación espectral (Burges, 2005), análisis clúster (Girolami, 2001) y un largo etcétera.

Esta rápida introducción a los métodos de detección de patrones mediante funciones *kernel* será complementada en los próximos apartados, que se dedicarán a la presentación detallada de cada uno de los elementos que componen estos procedimientos y, en particular, a la definición y presentación de las propiedades generales de las funciones *kernel*, a la descripción y enumeración de las propiedades de las funciones *kernel* más comúnmente empleadas, a la *kernelización* de ciertos algoritmos lineales, a la integración de las funciones *kernel* y los algoritmos *kernelizados* (posible gracias a la modularidad de este enfoque de aprendizaje estadístico) y a la aplicación de todo ello a algunos ejemplos prácticos.

2. Funciones *kernel*

Justificación del uso de las funciones kernel

El objetivo principal de proyectar o incrustar (*embed*) la base de datos original (de elementos del *input space* X) en el *feature space* F es modificar la representación del conjunto de datos original con el fin de facilitar la tarea de los algoritmos de búsqueda de patrones. Conviene recordar que, en definitiva, los algoritmos de detección de patrones basados en funciones *kernel* persiguen detectar relaciones de tipo lineal entre los datos transformados por el *embedding,* por lo que éste debería (idealmente) alterar la representación original de modo que sea sencillo identificar relaciones de tipo lineal en el conjunto de datos una vez que estos han sido transformados.

Para que el *embedding* sea eficaz y, en consecuencia, sea posible identificar patrones lineales relevantes en el conjunto de datos transformado, en ocasiones es necesario que la dimensión del *feature space* sea muy grande (incluso infinita). Esto constituye, a priori, un grave problema debido a la complejidad temporal cúbica de los procedimientos básicos del álgebra lineal que ya ha sido mencionada con anterioridad. Así, si la dimensión del *feature space* es muy elevada, el tiempo que el algoritmo necesitaría para encontrar un patrón sería inaceptable.

La conclusión se antoja descorazonadora ya que parecería que, si deseáramos dotar a los algoritmos de detección de patrones de complementos que les permitan superar las limitaciones inherentes al concepto de linealidad (entre ellos y, de forma destacada, su falta de flexibilidad), estaríamos abocados a tiempos de proceso inaceptables. Afortunadamente, esta conclusión es errónea. La solución que se ha encontrado a este problema es el empleo de las llamadas *funciones kernel*.

El razonamiento básico para la introducción de las funciones *kernel* es el siguiente: supongamos que disponemos de un conjunto de datos[11]

$$S=\{x_1,\ldots,x_l\} \quad \text{con} \quad x_i\in X \quad \forall i\in\{1,\ldots,l\} \quad \text{y} \quad X\subset\mathbb{R}^n$$

y que se ha definido una función (llamada función de *embedding*)

$$\phi: X\to F$$

que incrusta los elementos de S en un espacio de Hilbert y, por tanto, bien de dimensión infinita o bien isomorfo a $\mathbb{R}^N$ con $N\gg n$ (normalmente). A este espacio de Hilbert F se le da el nombre de *feature space.*

La función de *embedding* ha transformado el conjunto de datos original S (formado por vectores n-dimensionales- en otro conjunto de datos

$$\phi(S)=\{\phi(x_1),\ldots,\phi(x_l)\}$$

formado por vectores N-dimensionales.

11 Aunque aquí se supondrá que el *input space* es un subconjunto de $\mathbb{R}^n$, no es necesario que dicho espacio tenga una estructura algebraica concreta.

Una vez realizada la transformación del conjunto de datos mediante la función ϕ, el algoritmo de detección de patrones actuará sobre el *feature space* buscando relaciones de tipo lineal entre los elementos $\phi(x_i)$.

Supongamos, adicionalmente, que es posible "rediseñar" el algoritmo de detección de patrones de manera tal que para encontrar el patrón lineal en el *feature space* no sea necesario conocer cuáles son las imágenes por el *embedding* de los elementos del *input space* sino tan solo los productos escalares entre dichas imágenes. De manera más precisa, supongamos que el algoritmo de detección de patrones en F no precisa conocer $\phi(S)$ (que podría representarse como una matriz de l filas -los l elementos originales- y N columnas -la dimensión del *feature space,* posiblemente infinita-) sino que resulta suficiente -en el sentido estadístico del término- con conocer los módulos y las posiciones relativas en el *feature space* de los elementos de $\phi(S)$ (que es, en definitiva, la información que proporciona el producto escalar). Esa información acerca de los productos escalares entre las imágenes por el *embedding* de los elementos de X podría ser recogida en una matriz de l filas y l columnas (igual al número de elementos existentes en el conjunto de datos original e independiente tanto de la dimensión de X como de la dimensión de F). La posibilidad de este rediseño del algoritmo para que dependa sólo de los productos escalares y no de las coordenadas no es disparatada ya que, de hecho, gran parte de los algoritmos de detección de patrones lineales pueden ser "reprogramados" de esta manera (se dice que pueden ser *kernelizados*).

El hecho de que el algoritmo pueda ser reprogramado en la manera aludida solventaría el problema de la complejidad temporal de los algoritmos de búsqueda de patrones lineales pero dejaría aún pendientes de resolver las dificultades derivadas de la necesidad de incrustar (*embed*) los elementos del conjunto de datos original X en el *feature space* -de dimensión muy elevada o incluso infinita- para, posteriormente, calcular los productos escalares $<\phi(x_i), \phi(x_k)>$. En este sentido, resultaría del máximo interés que fuera posible determinar una función $k : X \times X \to \mathbb{R}$ que efectuara de manera simultánea y computacionalmente eficiente los dos pasos aludidos:

1) la transformación por el *embedding* de los datos de S en $\phi(S)$

2) el cálculo de los productos escalares de las imágenes $<\phi(x_i), \phi(x_k)>$

Está función estaría definida, para cada una de las posibles parejas formadas por elementos del conjunto S, del siguiente modo:

$$k(x_i, x_k) = <\phi(x_i), \phi(x_k)>$$

y proporcionaría directamente la información precisa para que el algoritmo de detección de patrones lineales (una vez *kernelizado*) alcanzara su objetivo. Se dice que la función k es la función *kernel* correspondiente al *embedding* $\phi : X \to F$.

A primera vista puede parecer que siempre es posible determinar la función k una vez conocidos el *embedding* ϕ y el producto escalar definido en el *feature space* ya que, al fin y al cabo, la función *kernel* k es simplemente el resultado de aplicar primero el *embedding* y luego calcular el producto escalar entre las imágenes. No obstante, el hecho de que en ocasiones la dimensión del *feature space* sea infinita hace que esta vía de actuación pueda resultar imposible. En cualquier caso, conviene razonar de modo inverso y plantearse la siguiente pregunta: ¿será posible partir de una función k, definida en $X \times X$ y con valores reales, y determinar un *feature space* F, que sea un espacio de Hilbert, y un *embedding* $\phi : X \to F$ de manera tal que

$k(x_i, x_k) = <\phi(x_i), \phi(x_k)>$ para todos los elementos de X ? La respuesta a esta pregunta es afirmativa (bajo ciertas condiciones que precisaremos).

Este nuevo planteamiento del problema nos proporciona un método distinto de trabajo. En efecto, si hemos afirmado:

- que la única alternativa viable para evitar el *overfitting* es la restricción de la familia de patrones que cabe esperar encontrar en el conjunto de datos original S y que esta restricción sólo cabe ser realizada a partir de la incorporación de conocimiento experto al procedimiento de aprendizaje.

- que la modularidad de los procedimientos de aprendizaje basados en funciones *kernel* hace que la incorporación de este conocimiento experto se produzca en la elección del *embedding*.

- que bajo ciertas condiciones (que precisaremos) siempre resulta posible, a partir de una determinada función $k : X \times X \to \mathbb{R}$ determinar un *embedding* y un *feature space* para los que la función k es la función *kernel*.

debemos concluir que la incorporación del conocimiento experto al procedimiento de aprendizaje basado en funciones *kernel* debe hacerse en la elección de dicha función *kernel* $k : X \times X \to \mathbb{R}$.

Así, una vez elegida la función $k : X \times X \to \mathbb{R}$ (que, tal como hemos anticipado, deberá cumplir ciertas características), estaremos en condiciones de afirmar que existe un *embedding* ϕ y un *feature space* F (que, obviamente, dependerán de la función *kernel* elegida y que, en muchas ocasiones, no tendremos interés en conocer pero asegurarán la aplicabilidad del método).

En cualquier caso, la función $k : X \times X \to \mathbb{R}$ permitirá el cálculo de los productos escalares (con el producto escalar definido en el espacio de Hilbert F) de las imágenes por el *embedding* de los elementos del conjunto S , es decir, proporcionará al algoritmo *kernelizado* toda la información que precisa para determinar patrones lineales en el *feature space,* que corresponden a patrones no lineales en el *input space* original.

Éste es, en esencia, el enfoque del aprendizaje estadístico basado en funciones *kernel*. Los siguientes apartados se dedican a formalizar estas cuestiones. Adicionalmente, los anexos M01 y M02 presentan los conceptos matemáticos básicos necesarios para la definición y caracterización de las funciones *kernel*.

Caracterización de las funciones kernel

El procedimiento que se ha propuesto hasta el momento para verificar que cierta función $k : X \times X \to \mathbb{R}$ es una función *kernel* (o un *kernel*, en breve) ha sido el siguiente:

- construir de forma explícita un *feature space* F (que debe ser un espacio de Hilbert)

- establecer, también explícitamente, una función de *embedding* $\phi : X \to F$

- comprobar que $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \; \forall \, x_i, x_j \in X$

Por ejemplo, si aplicamos este procedimiento a la función $k(x,z)=\langle x,z\rangle^2$ definida de $\mathbb{R}^2\times\mathbb{R}^2\to\mathbb{R}$ tendremos que si $x=(x_1\,x_2)^T$ y $z=(z_1\,z_2)^T$:

$$k(x,z)=\langle x,z\rangle^2=(x_1z_1+x_2z_2)^2=x_1^2z_1^2+x_2^2z_2^2+2x_1z_1x_2z_2=\langle(x_1^2,x_2^2,\sqrt{2}x_1x_2),(z_1^2,z_2^2,\sqrt{2}z_1z_2)\rangle$$

por lo que:

- el *feature space* F será $\mathbb{R}^3$, que con el producto escalar estándar es un espacio de Hilbert.
- el *embedding* será $\phi:\mathbb{R}^2\to\mathbb{R}^3:\phi(x_1,x_2)=(x_1^2,x_2^2,\sqrt{2}x_1x_2)$
- en consecuencia es inmediato comprobar que $k(x,z)=\langle\phi(x),\phi(z)\rangle\ \forall x,z\in\mathbb{R}^2$

Además de esta forma explícita de comprobar que una determinada función $k:X\times X\to\mathbb{R}$ es un *kernel* válido, expondremos a continuación un método alternativo que presenta la notable ventaja de prescindir de la construcción explícita de la función de *embedding* y del *feature space*. Para ello es necesario previamente definir el concepto de función *finitamente semidefinida positiva,* que será el que permitirá caracterizar un *kernel* valido.

Definición de función finitamente semidefinida positiva

Una función simétrica[12] $k:X\times X\to\mathbb{R}$ se dice que es *finitamente semidefinida positiva* si para todo subconjunto finito $S\subseteq X$, la restricción de la función k a $S\times S$ es una matriz semidefinida positiva.

Caracterización de las funciones *kernel*

Toda función simétrica $k:X\times X\to\mathbb{R}$ finitamente semidefinida positiva es un *kernel* válido. Efectivamente, dada una función simétrica finitamente semidefinida positiva $k:X\times X\to\mathbb{R}$ es posible definir un espacio de Hilbert F_k , dotado de un producto interno $\langle,\rangle:F_k\times F_k\to\mathbb{R}$ y una función $\phi:X\to F_k$ tales que $k(x,z)=\langle\phi(x),\phi(z)\rangle\ \forall x,z\in X$.

En efecto, consideremos un conjunto X y una función simétrica $k:X\times X\to\mathbb{R}$ finitamente semidefinida positiva. Nótese, en particular, que no se exige ningún tipo de estructura algebraica sobre el conjunto X .

A partir de estos elementos vamos a construir un espacio vectorial al que dotaremos de un producto interno. Completaremos ese espacio con algunos elementos con el fin de convertirlo en un espacio de Hilbert (dejaremos de lado la prueba de la separabilidad).

Comencemos por presentar los elementos que conforman ese espacio vectorial que, adelantémoslo, son funciones.

En efecto, en el conjunto X seleccionamos un subconjunto finito de l elementos $x_1,\ldots,x_l$ así como l números reales $\alpha_1,\ldots,\alpha_l$. Para cada uno de los x_i seleccionados en el conjunto X , consideramos $k(x_i,.)$ que, una vez fijado x_i , es una función $k(x_i,.):X\to\mathbb{R}$.

12 Una función real $k:X\times X\to\mathbb{R}$ es simétrica si $k(x,z)=k(z,x)\ \ \forall x,z\in X$

Construyamos ahora la función de X en $\mathbb{R}$ resultante de combinar linealmente todas las $k(x_i,.):X\to\mathbb{R}$ con los coeficientes reales $\alpha_1,\ldots,\alpha_l$. Se trata de la función:

$$\sum_{i=1}^{l}\alpha_i k(x_i,.):X\to\mathbb{R}$$

Estos son los elementos del espacio aludido.

En concreto, el conjunto F es:

$$F=\left\{\sum_{i=1}^{l}\alpha_i k(x_i,.):l\in\mathbb{N},x_i\in X,\alpha_i\in\mathbb{R},i=1,\ldots,l\right\}$$

que es un conjunto de funciones $f:X\to\mathbb{R}$

Ahora, dados dos elementos $f,g\in F$ definimos su suma $f+g$ del siguiente modo:

$$(f+g)(x)=f(x)+g(x)\quad\forall x\in X$$

Por otra parte, dado un elemento $f\in F$ y un escalar $\lambda\in\mathbb{R}$ definimos el producto de un elemento de F por el escalar λ del siguiente modo:

$$(\lambda f)(x)=\lambda f(x)\quad\forall x\in X$$

Resulta evidente que el conjunto F es cerrado a la suma y al producto por escalares, por lo que constituye un espacio vectorial.

A continuación, dotaremos al espacio vectorial F de un producto interno. Para ello, sobre el espacio vectorial F , vamos a definir la siguiente función $\langle,\rangle:F\times F\to\mathbb{R}$:

Dados dos elementos de F :

$$f=\sum_{i=1}^{l}\alpha_i k(x_i,.)\quad\text{y}\quad g=\sum_{j=1}^{n}\beta_j k(z_j,.)$$

entonces

$$\langle f,g\rangle=\sum_{i=1}^{l}\sum_{j=1}^{n}\alpha_i\beta_j k(x_i,z_j)=\sum_{i=1}^{l}\alpha_i g(x_i)=\sum_{j=1}^{n}\beta_j f(z_j)$$

Resulta evidente que la función $\langle,\rangle:F\times F\to\mathbb{R}$ es real, simétrica y bilineal.

Comprobaremos ahora que también es semidefinida positiva y concluiremos, en consecuencia, que la función $\langle,\rangle:F\times F\to\mathbb{R}$ es un producto interno en el espacio vectorial F .

En efecto, si tomamos cualquier $f=\sum_{i=1}^{l}\alpha_i k(x_i,.)\in F$ tenemos que:

$$\langle f , f \rangle = \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j k(x_i , x_j) = \alpha^T K \alpha$$

donde K es la matriz $l \times l$ que resulta de restringir la función $k : X \times X \to \mathbb{R}$ al subconjunto finito $x_1, \ldots, x_l$ de X. Pero como la función $k : X \times X \to \mathbb{R}$ es finitamente semidefinida positiva, resulta que la matriz K es semidefinida positiva y, en consecuencia $\langle f , f \rangle \geq 0$.

Así, la función $\langle , \rangle : F \times F \to \mathbb{R}$ definida por

$$\langle f , g \rangle = \sum_{i=1}^{l} \sum_{j=1}^{n} \alpha_i \beta_j k(x_i , z_j)$$

es un producto interno y el espacio vectorial F, dotado de este producto interno es un espacio con producto interno.

Este espacio con producto interno goza de la llamada *reproducing property*. En efecto, dado un elemento cualquiera $f \in F$ tenemos que:

$$\langle f , k(x , .) \rangle = \langle \sum_{i=1}^{n} \alpha_i k(x_i , .) , k(x , .) \rangle = \sum_{i=1}^{n} \alpha_i \langle k(x_i , .) , k(x , .) \rangle = \sum_{i=1}^{n} \alpha_i k(x_i , x) = f(x)$$

por lo que resulta que:

$$\langle f , k(x , .) \rangle = f(x) \quad \forall f \in F$$

Procederemos, por último, a completar el espacio con producto interno F con los límites de todas las sucesiones de Cauchy de elementos de F.

Para ello, consideremos un elemento fijo $x \in X$ y una sucesión de Cauchy de elementos de F $(f_n)_{n=1}^{\infty}$.

Resulta que:

$$[f_n(x) - f_m(x)]^2 = \langle f_n - f_m , k(x , .) \rangle^2 \leq \| f_n - f_m \|^2 \| k(x , .) \|^2 = \| f_n - f_m \|^2 k(x , x)$$

donde la primera igualdad es consecuencia de aplicar la *reproducing property* a $f_n - f_m$; la desigualdad, de la *desigualdad de Schwarz* y la segunda igualdad se cumple puesto que:

$$\| k(x , .) \|^2 = \langle k(x , .) , k(x , .) \rangle = k(x , x)$$

En resumen, la sucesión de Cauchy de elementos de $\mathbb{R}$, $(f_n(x))_{n=1}^{\infty}$ es acotada y, en consecuencia, tiene límite en $\mathbb{R}$.

Si definimos la función

$$g(x) = \lim_{n \to \infty} f_n(x)$$

y añadimos todas las funciones de esta clase al espacio con producto interno F, obtenemos un

espacio con producto interno completo al que llamaremos F_k, que será separable siempre que el conjunto X sea contable o la función k sea continua (véase, por ejemplo, Hein, 2005 para una demostración de esta propiedad).

En resumen, a partir de una función $k: X \times X \to \mathbb{R}$ finitamente semidefinida positiva hemos construido un espacio de Hilbert F_k.

Falta, por último, definir el *embedding* de X en F_k. Para ello, si consideramos la función $\phi: X \to F_k$ siguiente:

$$\phi(x) = k(x, .)$$

Se cumple que:

$$\langle \phi(x), \phi(z) \rangle = \langle k(x, .), k(z, .) \rangle = k(x, z)$$

para todo $x, z \in X$.

En resumen, hemos demostrado que es la propiedad de ser finitamente semidefinida positiva la que hace de una función $k: X \times X \to \mathbb{R}$ un *kernel* válido ya que, basándonos únicamente en esa propiedad, hemos sido capaces de determinar un espacio de Hilbert F_k y un *embedding* $\phi(x) = k(x, .)$ para los cuales $k: X \times X \to \mathbb{R}$ es una función *kernel*.

Debido a la *reproducing property* de que disfruta, se dice que el espacio de Hilbert F_k es un *Reproducing Kernel Hilbert Space.*

Conviene destacar, una vez más, que para la construcción del espacio de Hilbert y del *embedding* sólo se ha contado con dos elementos: el conjunto inicial X y la función *kernel*. En particular, no ha sido necesario definir ningún tipo de estructura sobre el conjunto X. En este sentido, resulta muy destacable la versatilidad que, de cara a la aplicación de algoritmos automáticos de detección de patrones, supone la utilización de funciones *kernel* ya que, como se ha visto, permiten la detección de patrones de naturaleza no lineal en conjuntos carentes de toda estructura algebraica.

Construcción de nuevos kernels a partir de otros

Hemos visto que la propiedad que sirve para caracterizar un *kernel* válido es su carácter de función real simétrica y finitamente semidefinida positiva.

Esta caracterización de los *kernels* permite razonar del siguiente modo: si combinamos funciones simétricas y finitamente semidefinidas positivas mediante operaciones que conserven dichas propiedades, estaremos obteniendo nuevos *kernels* válidos a partir de otros *kernels*. Es importante, en consecuencia, saber qué operaciones sobre funciones mantienen estas dos propiedades. Es lo que se consigue en el siguiente resultado:

Si k_1 y k_2 son *kernels* definidos en $X \times X$, con $X \subseteq \mathbb{R}^n$, a es un número real positivo, $f(\cdot)$ una función real definida en X, $\phi: X \to \mathbb{R}^N$, k_3 un *kernel* sobre $\mathbb{R}^N \times \mathbb{R}^N$ y B una matriz $n \times n$ simétrica y semidefinida positiva, entonces, las siguientes funciones son *kernels* válidos:

1. $k(x,z)=k_1(x,z)+k_2(x,z)$

2. $k(x,z)=a\,k_1(x,z)$

3. $k(x,z)=k_1(x,z)k_2(x,z)$

4. $k(x,z)=f(x)f(z)$

5. $k(x,z)=k_3(\phi(x),\phi(z))$

6. $k(x,z)=x^T B z$

La demostración de que cada una de estas nuevas funciones es, en realidad, un *kernel* válido se basa en comprobar el carácter de matriz simétrica y semidefinida positiva del resultado de restringir la evaluación de la función a un conjunto finito de X. Quizás merezcan un comentario adicional los casos 3 y 6.

- En el caso de la función $k(x,z)=k_1(x,z)k_2(x,z)$ se procede del siguiente modo. Elegimos un subconjunto finito $S=\{x_1,\ldots,x_l\}\subseteq X$ y evaluamos en S los *kernels* k_1 y k_2. El resultado son sendas matrices simétricas y semidefinidas positivas K_1 y K_2. La evaluación de $k(x,z)=k_1(x,z)k_2(x,z)$ es equivalente al *producto de Schur* (o *producto de Hadamard*) de las matrices K_1 y K_2. Basta aplicar el teorema que afirma que si A y B son matrices semidefinidas positivas, entonces el producto de Hadamard de A y B es también una matriz semidefinida positiva para concluir que $k(x,z)$ es un *kernel* válido. La simetría de esta matriz es evidente.

- El caso de la función $k(x,z)=x^T B z$ es ligeramente diferente y se apoya en el carácter de matriz semidefinida positiva y simétrica de B. En efecto, por ser simétrica y semidefinida positiva, la matriz B es ortogonalmente diagonalizable, con valores propios no negativos, es decir podemos escribir $B=V\Lambda V^T$. Ahora, como Λ es una matriz diagonal de términos no negativos, podemos descomponerla en el producto de dos matrices diagonales idénticas, con lo que $B=V\Lambda^{1/2}\Lambda^{1/2}V^T$. Si llamamos A a la matriz $\Lambda^{1/2}V^T$ tenemos que $B=A^T A$ y $x^T B z=x^T A^T Az=\langle Ax, Az\rangle$ con lo que $k(x,z)=x^T B z$ es el *kernel* resultante de calcular el producto escalar en el *feature space* correspondiente una vez que se ha llevado a cabo el *embedding* asociado a la matriz $A=\Lambda^{1/2}V^T$.

En cualquier caso, se pone de manifiesto que una forma sencilla de obtener nuevas funciones *kernel* es mediante la combinación (mediante operaciones que conserven la simetría y la propiedad de ser finitamente semidefinida positiva) de otras funciones *kernel*. Utilizaremos estas propiedades para demostrar el siguiente resultado, que va a permitir definir algunas funciones *kernel* muy comúnmente utilizadas: el *kernel polinómico* y el *kernel gaussiano*.

Sea $k_1(x,z):X\times X\to\mathbb{R}$ un *kernel* válido y $p(x)$ un polinomio con coeficientes positivos. Entonces, las siguientes funciones también son *kernels* válidos:

1. $k(x,z)=p[k_1(x,z)]$ (*kernel polinómico*)

2. $k(x,z)=\exp[k_1(x,z)]$

3. $k(x,z)=\exp[-\|x-z\|^2/(2\sigma^2)]$ (*kernel gaussiano*)

La demostración de este resultado es inmediata:

- En cuanto a $k(x,z)=p[k_1(x,z)]$, sabemos que el producto de un *kernel* por otro, el producto de *kernels* por un número real positivo y la suma de *kernels* proporcionan como resultado *kernels* válidos, por lo que se deduce que $k(x,z)=p[k_1(x,z)]$ es un *kernel* válido.

- La demostración de que $k(x,z)=\exp(k_1(x,z))$ es un *kernel* válido se apoya en el conocido hecho de que el desarrollo de Taylor de $\exp(x)$ es:

$$\exp(x)=\sum_{i=0}^{\infty}\frac{x^i}{i!}$$

  por lo que la función $\exp(x)$ puede aproximarse cuanto se desee por una sucesión de polinomios con coeficientes positivos. Como el carácter de ser una función finitamente semidefinida positiva se conserva en el paso al límite, se deduce que

$$k(x,z)=\exp(k_1(x,z))$$

  es un *kernel* válido.

- La demostración del tercer resultado parte del hecho de que $\langle x,z\rangle/\sigma^2$ es un *kernel* válido. En consecuencia, por el resultado anterior tenemos que $\exp(\langle x,z\rangle/\sigma^2)$ es un *kernel* válido. Por otra parte, dado un *kernel* cualquiera $k(x,z)$, podemos *normalizar* este *kernel* mediante la siguiente operación:

$$\hat{k}(x,z)=\frac{k(x,z)}{\sqrt{k(x,x)k(z,z)}}$$

  Es evidente que, siempre que $k(x,x)\neq 0\ \forall x\in X$, $\hat{k}(x,z)$ es un *kernel* válido. En efecto, dado un conjunto finito $S=\{x_1,\ldots,x_l\}$ de X, la matriz asociada al *kernel* sin normalizar $k(x,z)$ y al conjunto S será una matriz simétrica y semidefinida positiva K. Los elementos de la diagonal principal de dicha matriz serán todos positivos (por ser de la forma $k(x_j,x_j)$) y, por lo tanto, será posible construir una matriz $D^{-1/2}$ cuyos componentes serán de la forma $k(x_j,x_j)^{-1/2}$. La matriz asociada al *kernel* normalizado y al conjunto S será $D^{-1/2}KD^{-1/2}$. Además, por ser K simétrica y semidefinida positiva, es ortogonalmente diagonalizable con valores propios no negativos, es decir, existen una matriz ortogonal U (cuyas columnas son vectores propios y unitarios de K) y una matriz diagonal Λ con elementos no negativos tales que:

$$K=U\Lambda U^T=U\Lambda^{1/2}\Lambda^{1/2}U^T$$

  Por todo ello, la matriz $\hat{K}$ asociada al conjunto S y al *kernel* normalizado será:

$$\hat{K} = D^{-1/2} K D^{-1/2} = D^{-1/2} U \Lambda^{1/2} \Lambda^{1/2} U^T D^{-1/2} = (\Lambda^{1/2} U^T D^{-1/2})^T (\Lambda^{1/2} U^T D^{-1/2})$$

que, por ser de la forma $B^T B$, es simétrica y semidefinida positiva.

En consecuencia, si normalizamos el *kernel* $\exp(\langle x, z\rangle/\sigma^2)$ obtenemos:

$$\frac{\exp(\langle x, z\rangle/\sigma^2)}{\sqrt{\exp(\langle x, x\rangle/\sigma^2)\exp(\langle z, z\rangle/\sigma^2)}} = \exp\left(\frac{\langle x, z\rangle}{\sigma^2} - \frac{\langle x, x\rangle}{2\sigma^2} - \frac{\langle z, z\rangle}{2\sigma^2}\right) = \exp\left(-\|x-z\|^2/2\sigma^2\right)$$

donde se ha tenido en cuenta que:

$$\|x-z\|^2 = \langle x-z, x-z\rangle = (x-z)^T(x-z) = x^T x - x^T z - z^T x + z^T z = \langle x, x\rangle + \langle z, z\rangle - 2\langle x, z\rangle$$

Dos funciones kernel muy habituales: el kernel polinómico y el kernel gaussiano

Tras presentar las propiedades básicas de las funciones *kernel,* su caracterización como funciones simétricas y finitamente semidefinidas positivas y la posibilidad de construir nuevas funciones *kernel* a partir de otras, dedicaremos este apartado a la presentación de las características básicas de dos *kernels* muy frecuentemente utilizados: el *kernel polinómico* y el *kernel gaussiano*.

El *kernel* polinómico

En el anterior apartado hemos demostrado que si $k(x, z)$ es un *kernel* válido y $p(\cdot)$ un polinomio con coeficientes positivos, entonces $p[k(x, z)]$ es un *kernel* válido al que hemos denominado *kernel polinómico*.

No obstante, cuando se habla del *kernel polinómico*, es habitual que se esté haciendo referencia a un caso particular de este *kernel* polinómico en sentido amplio. En concreto, dado un *input space* X contenido en $\mathbb{R}^n$ definiremos el *kernel polinómico de grado* d *y offset* R del siguiente modo:

$$k_d(x, z) = (\langle x, z\rangle + R)^d \qquad \forall\, x, z \in X$$

Es evidente que este *kernel* polinómico así definido encaja plenamente en la definición de *kernel* polinómico más amplia. En efecto, el *kernel* válido inicial sería $\langle x, z\rangle$ (lo que exige que el *input space* X sea un espacio con producto interno) y el polinomio con coeficientes positivos sería $(x+R)^d$.

Vamos a determinar cuál es el *feature space* y el *embedding* que resultan de la elección de un *kernel* polinómico como el que se acaba de definir. Para ello debemos tomar en consideración que $x, z \in X \subseteq \mathbb{R}^n$, por lo que:

$$\langle x, z\rangle = \sum_{i=1}^{n} x_i z_i$$

y

$$(\langle x, z\rangle + R)^d = \left(\sum_{i=1}^{n} x_i z_i + R\right)^d$$

Desarrollando la expresión anterior tenemos que:

$$(\langle x, z\rangle + R)^d = \sum_{i_0+\ldots+i_n=d} \frac{d!}{i_0!\ldots i_n!} R^{i_0} x_1^{i_1} z_1^{i_1} \ldots x_n^{i_n} z_n^{i_n}$$

con i_j entero tal que $0 \le i_j \le d \quad \forall\, j \in \{0,\ldots,n\}$

Para conocer cuál es el número de sumandos de la expresión anterior tenemos que responder a la siguiente pregunta: ¿de cuántas formas se puede obtener el resultado d mediante la suma de $n+1$ enteros no negativos? La respuesta es:

$$\binom{n+d}{d} = \frac{(n+d)!}{d!\,n!}$$

Por otra parte, consideremos la función

$$\phi : X \to \mathbb{R}^{\binom{n+d}{d}}$$

que a cada elemento $x \in X \subseteq \mathbb{R}^n$ le hace corresponder $\phi(x)$, que es un vector cuyos $\binom{n+d}{d}$ componentes son de la forma

$$\left(\frac{d!}{i_0!\ldots i_n!} R^{i_0}\right)^{1/2} x_1^{i_1} \ldots x_n^{i_n}$$

siendo $\{i_0, i_1, \ldots i_n\}$ uno de los $\binom{n+d}{d}$ conjuntos de $n+1$ enteros no negativos que cumplen que $0 \le i_j \le d \quad \forall\, j \in \{0,\ldots,n\}$ y que $i_0 + i_1 + \ldots + i_n = d$.

A partir de estas definiciones resulta que:

$$(\langle x, z\rangle + R)^d = \sum_{i_0+\ldots+i_n=d} \left[\left(\sqrt{\frac{d!}{i_0!\ldots i_n!} R^{i_0}}\right) x_1^{i_1} \ldots x_n^{i_n}\right]\left[\left(\sqrt{\frac{d!}{i_0!\ldots i_n!} R^{i_0}}\right) z_1^{i_1} \ldots z_n^{i_n}\right]$$

En definitiva, el *kernel* polinómico $k_d(x, z) = (\langle x, z\rangle + R)^d$ induce:

- Una función de *embedding* ϕ de $X \subseteq \mathbb{R}^n$ en $\mathbb{R}^{\binom{n+d}{d}}$ que a cada elemento $x \in X$ le hace corresponder un vector $\phi(x)$ cuyos componentes son de la forma:

$$\left(\frac{d!}{i_0!\ldots i_n!}R^{i_0}\right)^{1/2}x_1^{i_1}\ldots x_n^{i_n}$$

- Un *feature space* $\mathbb{R}^{\binom{n+d}{d}}$ con el producto escalar estándar y cuyas dimensiones están indexadas por todas las expresiones de la forma

$$x_1^{i_1}\ldots x_n^{i_n}$$

es decir, por todos los *monomios de grado menor o igual que* d (téngase en cuenta que en la expresión anterior se ha omitido el indice i_0), que constituyen los patrones que cabe localizar en el *input space* X .

Finalmente, conviene destacar que el parámetro R (llamado *offset)* permite ajustar el peso relativo de cada uno de los monomios. En efecto, a partir de la expresión

$$\left(\frac{d!}{i_0!\ldots i_n!}R^{i_0}\right)^{1/2}x_1^{i_1}\ldots x_n^{i_n}$$

vemos que cuanto mayor sea i_0 y, en consecuencia, cuanto menor sea el grado del monomio $x_1^{i_1}\ldots x_n^{i_n}$ mayor será la influencia de R en el valor de la expresión anterior. En definitiva, un valor elevado de R proporciona un elevado peso relativo a los monomios de grado reducido y, por tanto, un menor peso relativo a los monomios de grado elevado.

El *kernel* gaussiano

En la sección dedicada a la presentación y caracterización de las funciones *kernel* en general se presentó el caso del siguiente *kernel* (al que se denominó *kernel gaussiano*):

$$k(x,z)=\exp[-\|x-z\|^2/(2\sigma^2)]$$

Su validez como función *kernel* se justificó a partir de tres hechos:

- El desarrollo en serie de Taylor de la función exponencial
- La conservación por el paso al límite de la propiedad de una función simétrica de ser finitamente semidefinida positiva.
- La normalización de una función *kernel*

En este apartado pasaremos revista a algunas de las propiedades más destacables del *kernel* gaussiano que, por su flexibilidad, es uno de los más frecuentemente utilizados:

- Las imágenes por el *embedding* inducido por el *kernel gaussiano* son vectores unitarios. En efecto:

$$\|\phi(x)\|^2=\langle\phi(x),\phi(x)\rangle=k(x,x)=\exp(0)=1$$

- Dados dos elementos x, z distintos de X, el ángulo que forman sus imágenes por el *embedding* en el *feature space* es un ángulo agudo. Efectivamente, por una parte:

$$k(x,z)=\exp[-\|x-z\|^2/(2\sigma^2)]\leq 1$$

y por otra

$$k(x,x)=k(z,z)=1$$

En consecuencia,

$$\cos(\phi(x),\phi(z))=\frac{\langle\phi(x),\phi(z)\rangle}{\sqrt{\langle\phi(x),\phi(x)\rangle\langle\phi(z),\phi(z)\rangle}}=\frac{k(x,z)}{\sqrt{k(x,x)k(z,z)}}\leq 1$$

- Esto implica que el *feature space* puede elegirse de manera tal que todas las imágenes de los elementos del *input space* se encuentren en un hiperoctante[13].

- El parámetro σ del *kernel gaussiano* permite controlar la flexibilidad del *kernel*. Si el parámetro σ es muy reducido, la función *kernel* asignará valores muy cercanos a cero a las parejas formadas por elementos distintos del *input space* (aunque su diferencia sea muy pequeña). De esta forma, la matriz *kernel* será muy parecida a la matriz identidad y se caerá en el problema de *overfitting*. En el otro extremo, valores muy elevados de σ tendrán como consecuencia que los valores asignados por la función *kernel* a las parejas de elementos serán muy cercanos a 1 (sean o no parecidos). La función *kernel* será, en estos casos, una función prácticamente constante y no será de utilidad para la detección de patrones.

- No es sencillo visualizar cuál es el *feature space* y el *embedding* inducidos por el *kernel* gaussiano aunque siempre cabe utilizar el teorema que afirma que dado cualquier *kernel* válido podemos definir un *feature space* y un *embedding* asociados con él. Aplicando dicho teorema tenemos que el *feature space* asociado al *kernel* gaussiano está formado por todas las funciones de la forma

$$f=\sum_{i=1}^{l}\alpha_i k(x_i,.)=\sum_{i=1}^{l}\alpha_i\exp(-\|x_i-\cdot\|^2/(2\sigma^2))$$

donde $\{x_1,\ldots,x_l\}$ es un conjunto finito de elementos del *input space* X y los $\alpha_1,\ldots,\alpha_l$ son números reales. En cuanto al *embedding* resulta ser:

$$\phi(x)=\exp(-\|x-\cdot\|^2/(2\sigma^2))$$

Puede comprobarse que la dimensión del *feature space* asociado al *kernel* gaussiano es infinita.

Una vez revisadas las características básicas de las funciones *kernel*, establecida su caracterización como funciones reales simétricas y finitamente semidefinidas positivas y presentadas las características básicas de los *kernels* polinómico y gaussiano, dedicaremos la sección siguiente a la

13 Un hiperoctante es la generalización n dimensional de lo que en el espacio $\mathbb{R}^2$ es un cuadrante.

presentación de una batería de procedimientos para extraer información de la matriz *kernel*.

3. Algunos algoritmos elementales en el *feature space*

A lo largo de todo este apartado consideramos un subconjunto finito $S=\{x_1,\ldots,x_l\}$ de puntos de X, que constituye un subconjunto del *input space,* y una función *kernel* $k:X\times X\to\mathbb{R}$ que, evaluada sobre S, proporciona la matriz *kernel* K, de dimensión $l\times l$.

Nos proponemos conocer qué información sobre $\phi(S)=\{\phi(x_1),\ldots,\phi(x_l)\}$ se puede extraer a partir, exclusivamente, de la matriz K. Veremos que, contando únicamente con la matriz *kernel,* es posible conocer:

- La norma de los elementos de $\phi(S)$
- La norma de una combinación lineal de elementos de $\phi(S)$
- La distancia entre dos elementos cualesquiera de $\phi(S)$
- La norma del centro de masas de $\phi(S)$
- La distancia entre un elemento de $\phi(S)$ y el centro de masas
- La distancia promedio entre los elementos de $\phi(S)$ y su centro de masas

Terminaremos presentando dos procedimientos que serán de utilidad más adelante y que, en ambos casos, emplean únicamente la información procedente de la matriz *kernel*:

- Un procedimiento para el centrado de los elementos de $\phi(S)$ en el *feature space*.
- El cálculo de la variabilidad de las proyecciones de los elementos de $\phi(S)$ a lo largo de una dirección determinada del *feature space*.

Norma de las imágenes de los elementos de S

La norma de los elementos del conjunto $\phi(S)$ puede obtenerse directamente, calculando las raíces cuadradas de los elementos que ocupan la diagonal principal de la matriz *kernel* K. En efecto:

$$\|\phi(x)\|=\sqrt{\langle\phi(x),\phi(x)\rangle}=\sqrt{k(x,x)}$$

Norma de una combinación lineal de los elementos de $\phi(S)$

Dada una combinación lineal de elementos de $\phi(S)$:

$$\sum_{i=1}^{l}\alpha_i\phi(x_i)$$

el cuadrado de la norma de dicha combinación lineal se puede obtener a través de la matriz K y resulta ser $\alpha^T K \alpha$, siendo α el vector de dimensión $l \times 1$ que recoge los coeficientes de la combinación lineal.

En efecto:

$$\left\| \sum_{i=1}^{l} \alpha_i \phi(x_i) \right\|^2 = \left\langle \sum_{i=1}^{l} \alpha_i \phi(x_i), \sum_{j=1}^{l} \alpha_j \phi(x_j) \right\rangle = \sum_{i,j=1}^{l} \alpha_i \alpha_j \langle \phi(x_i), \phi(x_j) \rangle = \sum_{i,j=1}^{l} \alpha_i \alpha_j k(x_i, x_j) = \alpha^T K \alpha$$

Distancia entre dos elementos de $\phi(S)$

La distancia entre dos elementos de $\phi(S)$ también se puede obtener directamente a partir de los elementos de la matriz K .

Sabemos que:

$$\|\phi(x) - \phi(z)\|^2 = \langle \phi(x) - \phi(z), \phi(x) - \phi(z) \rangle$$

pero

$$\langle \phi(x) - \phi(z), \phi(x) - \phi(z) \rangle = \langle \phi(x), \phi(x) \rangle - 2\langle \phi(x), \phi(z) \rangle + \langle \phi(z), \phi(z) \rangle$$

por lo que

$$\|\phi(x) - \phi(z)\|^2 = k(x,x) - 2k(x,z) + k(z,z)$$

Se trata, en realidad, de un caso particular del caso anterior (norma de una combinación lineal de los elementos de $\phi(S)$) en el que el vector α , que recoge los coeficientes de la combinación lineal, tiene todos sus elementos nulos salvo el correspondiente al elemento x (cuyo valor es 1) y el correspondiente a z (que tiene valor -1). Así:

$$\|\phi(x) - \phi(z)\|^2 = \begin{pmatrix} 0 & \dots & 1 & 0 & \dots & 0 & -1 & 0 & \dots \end{pmatrix} K \begin{pmatrix} 0 \\ \dots \\ 1 \\ 0 \\ \dots \\ 0 \\ -1 \\ 0 \\ \dots \end{pmatrix} = k(x,x) - 2k(x,z) + k(z,z)$$

Norma del centro de masas de $\phi(S)$

Si denotamos por Φ_S al centro de masas de $\phi(S)$ tenemos que:

$$\Phi_S = \frac{1}{l}\sum_{i=1}^{l}\phi(x_i)$$

es decir, el centro de masas de $\phi(S)$ es una combinación lineal de los elementos de $\phi(S)$ cuyos coeficientes son todos iguales a $1/l$ y, por tanto, el cuadrado de su norma será:

$$\|\Phi_S\|^2 = \begin{pmatrix}\frac{1}{l} & \dots & \frac{1}{l}\end{pmatrix} K \begin{pmatrix}\frac{1}{l} \\ \dots \\ \frac{1}{l}\end{pmatrix} = \frac{1}{l^2}\sum_{i,j=1}^{l} k(x_i, x_j)$$

es decir, el promedio de los elementos de la matriz K.

Por ser K una matriz semidefinida positiva (y por la ecuación anterior), resulta que $\|\Phi_S\|^2 \geq 0$ siendo $\|\Phi_S\|^2 = 0$ cuando Φ_S coincide con el origen del *feature space*.

Distancia de un elemento de $\phi(S)$ *al centro de masas de* $\phi(S)$

El cuadrado de la distancia de un elemento de $\phi(x) \in \phi(S)$ al centro de masas Φ_S es:

$$\|\phi(x) - \Phi_S\|^2 = \langle \phi(x), \phi(x) \rangle - 2\langle \phi(x), \Phi_S \rangle + \langle \Phi_S, \Phi_S \rangle$$

pero

$$\langle \phi(x), \Phi_S \rangle = \left\langle \phi(x), \frac{1}{l}\sum_{i=1}^{l}\phi(x_i) \right\rangle = \frac{1}{l}\sum_{i=1}^{l}\langle \phi(x), \phi(x_i) \rangle$$

luego

$$\|\phi(x) - \Phi_S\|^2 = k(x,x) - \frac{2}{l}\sum_{i=1}^{l} k(x, x_i) + \frac{1}{l^2}\sum_{i,j=1}^{l} k(x_i, x_j)$$

Así, el cuadrado de la distancia de un elemento de $\phi(S)$ al centro de masas Φ_S viene dado por tres sumandos:

- $k(x,x)$: es decir, el elemento de la diagonal principal de la matriz K que corresponde a x .
- $\sum_{i=1}^{l} k(x, x_i)$, es decir, la suma de los elementos de la fila de la matriz K

correspondiente al elemento x .

- $\sum_{i,j=1}^{l} k(x_i, x_j)$, es decir, la suma de todos los elementos de la matriz K .

Podemos verlo también como la norma de una combinación lineal de elementos de $\phi(S)$ en la que el vector α que recoge los coeficientes de la combinación es de la forma:

$$\alpha^T = \left[\frac{-1}{l} \quad \dots \quad \left(1 - \frac{1}{l}\right) \quad \frac{-1}{l} \quad \dots \right]$$

donde el elemento de la forma $1 - \frac{1}{l}$ es el correspondiente al elemento x . De esta forma tendremos, igualmente, que:

$$\|\phi(x) - \phi_S\|^2 = \alpha^T K \alpha = k(x,x) - \frac{2}{l}\sum_{i=1}^{l} k(x, x_i) + \frac{1}{l^2}\sum_{i,j=1}^{l} k(x_i, x_j)$$

Distancia al cuadrado promedio de las imágenes de los elementos de S al centro de masas de $\phi(S)$

Aplicando la fórmula anterior se obtiene fácilmente que:

$$\frac{1}{l}\sum_{m=1}^{l} \|\phi(x_m) - \phi_S\|^2 = \frac{1}{l}\sum_{m=1}^{l} k(x_m, x_m) - \frac{2}{l}\frac{1}{l}\sum_{m=1}^{l}\sum_{i=1}^{l} k(x_m, x_i) + \frac{1}{l^2}\frac{1}{l}\sum_{m=1}^{l}\sum_{i,j=1}^{l} k(x_i, x_j)$$

Pero:

$$\sum_{m=1}^{l}\sum_{i,j=1}^{l} k(x_i, x_j) = l \sum_{i,j=1}^{l} k(x_i, x_j)$$

y, en consecuencia:

$$\frac{1}{l^2}\frac{1}{l}\sum_{m=1}^{l}\sum_{i,j=1}^{l} k(x_i, x_j) = \frac{1}{l^2}\sum_{i,j=1}^{l} k(x_i, x_j)$$

y operando de modo elemental:

$$\frac{1}{l}\sum_{m=1}^{l} \|\phi(x_m) - \phi_S\|^2 = \frac{1}{l}\sum_{m=1}^{l} k(x_m, x_m) - \frac{1}{l^2}\sum_{i,j=1}^{l} k(x_i, x_j)$$

es decir, el promedio de los cuadrados de las distancias de los elementos de $\phi(S)$ al centro de masas de $\phi(S)$ es igual al promedio de los elementos de la diagonal principal de la matriz K menos el promedio del conjunto de elementos de la matriz K .

Sabemos, además, que si los datos se encuentran centrados en el *feature space*, entonces:

$$\|\phi_S\|^2 = \frac{1}{l^2}\sum_{i,j=1}^{l} k(x_i, x_j) = 0$$

En consecuencia:

$$\frac{1}{l}\sum_{m=1}^{l} \|\phi(x_m) - \phi_S\|^2 = \frac{1}{l}\sum_{m=1}^{l} k(x_m, x_m) = \frac{1}{l} tr(K)$$

de donde se concluye que la traza de la matriz K es una medida de la variabilidad de los elementos de $\phi(S)$ respecto a su centro de gravedad.

Algoritmo para el centrado de datos en el feature space

Para muchas aplicaciones resulta interesante centrar los datos en el *feature space* es decir, trasladar ϕ_S al origen de F .

Tras realizar dicha operación de centrado ocurre que:

- $\frac{1}{l}\sum_{m=1}^{l} \|\phi(x_m) - \phi_S\|^2 = \frac{1}{l}\sum_{m=1}^{l} k(x_m, x_m) - \frac{1}{l^2}\sum_{i,j=1}^{l} k(x_i, x_j)$ permanece constante ya que el centrado es una traslación y, en consecuencia, preserva las distancias entre los elementos del espacio.

- $\|\phi_S\|^2 = \frac{1}{l^2}\sum_{i,j=1}^{l} k(x_i, x_j) = 0$

En consecuencia, al trasladar ϕ_S al origen del *feature space*, se minimiza el valor de

$$\|\phi_S\|^2 = \frac{1}{l^2}\sum_{i,j=1}^{l} k(x_i, x_j)$$

y, en consecuencia, también se hace mínimo el valor de:

$$\frac{1}{l}\sum_{m=1}^{l} k(x_m, x_m) = \frac{1}{l}\sum_{m=1}^{l} \|\phi(x_m)\|^2$$

En resumen, al trasladar el centro de masas de $\phi(S)$ al origen del *feature space* se minimiza la suma de los cuadrados de las normas de los elementos de $\phi(S)$. Es decir:

$$\phi_S = argmin_u \sum_{m=1}^{l} \|\phi(x_m) - u\|^2$$

Este resultado no es sino una generalización multidimensional del hecho de que la media aritmética es el punto respecto al cual se hace mínimo el momento de segundo orden de un conjunto de datos.

$$\bar{x} = argmin_k \sum_{i=1}^{l} (x_i - k)^2$$

Adicionalmente, conviene señalar que el valor de la función minimizada en el óptimo es precisamente:

$$\frac{1}{l}\sum_{m=1}^{l} \|\phi(x_m)\|^2 = \frac{1}{l}\sum_{m=1}^{l} k(x_m, x_m) = \frac{1}{l} tr(K)$$

es decir, la traza de la matriz K dividida entre su dimensión. Como la traza de una matriz simétrica es la suma de sus valores propios, podemos concluir que el valor minimizado es el valor propio promedio de la matriz K. Dicho más claramente: trasladar el centro de masas de $\phi(S)$ al origen del *feature space* hace mínimo el valor propio promedio de la matriz *kernel*. Tampoco este resultado debe ser extraño ya que la traza de una *matriz de inercia* (y la matriz *kernel* lo es) recoge la variabilidad de los individuos respecto a cierto punto del espacio que, en este caso y por estar los datos centrados, es el origen de coordenadas.

Veamos cómo afecta la operación de centrado a la función *kernel* y, en consecuencia, a la matriz *kernel*. Tras centrar los datos, encontramos que el nuevo *embedding* es:

$$\hat{\phi}(x) = \phi(x) - \phi_S = \phi(x) - \frac{1}{l}\sum_{i=1}^{l} \phi(x_i)$$

y la función *kernel* asociada a él:

$$\hat{k}(x, z) = \langle \hat{\phi}(x), \hat{\phi}(z) \rangle = \left\langle \phi(x) - \frac{1}{l}\sum_{i=1}^{l} \phi(x_i), \phi(z) - \frac{1}{l}\sum_{i=1}^{l} \phi(x_i) \right\rangle$$

es decir,

$$\hat{k}(x, z) = \langle \phi(x), \phi(z) \rangle - \frac{1}{l}\sum_{i=1}^{l} \langle \phi(x), \phi(x_i) \rangle - \frac{1}{l}\sum_{i=1}^{l} \langle \phi(x_i), \phi(z) \rangle + \frac{1}{l^2}\sum_{i,j=1}^{l} \langle \phi(x_i), \phi(x_j) \rangle$$

y, en definitiva:

$$\hat{k}(x, z) = k(x, z) - \frac{1}{l}\sum_{i=1}^{l} k(x, x_i) - \frac{1}{l}\sum_{i=1}^{l} k(z, x_i) + \frac{1}{l^2}\sum_{i,j=1}^{l} k(x_i, x_j)$$

En resumen, hemos encontrado una fórmula para transformar la matriz *kernel* K en la matriz *kernel* $\hat{K}$ correspondiente a la operación de centrado de datos. Para obtener un elemento de la matriz $\hat{K}$ debemos:

- Seleccionar el elemento $k(x, z)$ de la matriz K
- Restar a ese elemento el promedio de los elementos que están en la misma fila:

$$\frac{1}{l}\sum_{i=1}^{l} k(x, x_i)$$

- Restar a ese elemento el promedio de los elementos que se encuentran en la misma columna:

$$\frac{1}{l}\sum_{i=1}^{l} k(z, x_i) = \frac{1}{l}\sum_{i=1}^{l} k(x_i, z)$$

- Sumar el promedio de los elementos de la matriz K :

$$\frac{1}{l^2}\sum_{i,j=1}^{l} k(x_i, x_j)$$

Matricialmente, si denotamos por j un vector columna de unos de dimensión $l \times 1$:

- $K\,j$ es un vector columna $l \times 1$ que contiene las sumas de las filas (y las columnas, ya que la matriz es simétrica) de la matriz K

- $K\,j\,j^T$ es una matriz $l \times l$ cuyas columnas contienen las sumas de las filas (y las columnas) de la matriz K

- $j\,j^T K$ es una matriz $l \times l$ cuyas filas contienen las sumas de las filas (y las columnas) de la matriz K

- $j^T K\,j\,j\,j^T$ es una matriz $l \times l$ cuyos elementos son todos iguales a la suma de los elementos de la matriz K

En consecuencia:

$$\hat{K} = K - \frac{1}{l} K\,j\,j^T - \frac{1}{l} j\,j^T K + \frac{1}{l^2} j^T K\,j\,j\,j^T$$

En definitiva, hemos obtenido un método que permite centrar los datos en el *feature space* mediante una simple manipulación de la matriz *kernel* K . Veremos, más adelante, que este método resultará de utilidad en el desarrollo del algoritmo conocido como *kernel PCA*.

Varianza de las proyecciones de las imágenes de los elementos de S en una dirección del feature space

Consideremos un conjunto de elementos en el *input space* $S = \{x_1, \ldots, x_l\}$ con $S \subset \mathbb{R}^n$ y un *embedding* $\phi : \mathbb{R}^n \to \mathbb{R}^N$, de manera que el *feature space* es N -dimensional.

Consideremos, a continuación, la matriz X de dimensión $l \times N$ cuyas filas son las imágenes por el *embedding* de los elementos x_i , es decir, la *i*-ésima fila de la matriz X es $\phi(x_i)^T$.

Supongamos, además, que se ha procedido a centrar los datos en el *feature space*, de manera que

ϕ_S coincide con el origen de $\mathbb{R}^N$.

Bajo estas condiciones, la covarianza entre las coordenadas de los elementos de $\phi(S)$ en dos dimensiones s, t cualesquiera del *feature space* se define del modo siguiente:

$$\frac{1}{l}\sum_{i=1}^{l} X_{is} X_{it} = \frac{1}{l}\sum_{i=1}^{l} \phi(x_i)_s \phi(x_i)_t$$

La matriz de covarianzas C entre las dimensiones del *feature space* es la matriz $N \times N$ cuyo elemento genérico C_{st} es la covarianza entre las dimensiones s y t . En consecuencia:

$$C = \frac{1}{l} X^T X$$

En el desarrollo de ciertas tareas de análisis de datos puede resultar interesante analizar la dispersión de las proyecciones de los elementos del *feature space* en una dirección de dicho espacio. Si dicha dirección viene expresada por un vector unitario, al que denotaremos por $v \in \mathbb{R}^N$, podemos calcular la proyección de un elemento cualquiera del *feature space* sobre el vector v del siguiente modo:

$$P_v(\phi(x)) = \left[v^T \phi(x)\right] v = \left[\phi(x)^T v\right] v$$

En consecuencia, el cuadrado del módulo de la proyección será:

$$\|P_v(\phi(x))\|^2 = (v^T \phi(x) v)^T (v^T \phi(x) v) = v^T \phi(x)^T v \, v^T \phi(x) v$$

pero

$$\phi(x)^T v \, v^T \phi(x) = [v^T \phi(x)]^T [v^T \phi(x)]$$

es un escalar y, en consecuencia:

$$\|P_v(\phi(x))\|^2 = [v^T \phi(x)]^T [v^T \phi(x)] v^T v = [v^T \phi(x)]^T [v^T \phi(x)] = [v^T \phi(x)]^2$$

ya que, por ser v un vector unitario

$$v^T v = \|v\|^2 = 1$$

de donde se deduce que:

$$\|P_v(\phi(x))\| = v^T \phi(x)$$

Si suponemos (como hemos hecho) que los datos están centrados en el *feature space*, resultará que:

$$\frac{1}{l}\sum_{i=1}^{l} \|P_v(\phi(x_i))\| = v^T \frac{1}{l}\sum_{i=1}^{l} \phi(x_i) = v^T \phi_S = 0$$

y, en consecuencia, la varianza de los módulos de las proyecciones de los elementos de $\phi(S)$ a

lo largo del vector v será:

$$\frac{1}{l}\sum_{i=1}^{l}\|P_v(\phi(x_i))\|^2=\frac{1}{l}\sum_{i=1}^{l}[v^T\phi(x_i)]^2=\frac{1}{l}\left[v^T\phi(x_1)\quad\ldots\quad v^T\phi(x_l)\right]\begin{bmatrix}\phi(x_1)^T v\\ \ldots\\ \phi(x_l)^T v\end{bmatrix}=\frac{1}{l}v^T\left(X^T X\right)v$$

Si elegimos el vector unitario v como una combinación lineal de los elementos de $\phi(S)$, es decir, si:

$$v=\alpha_1\phi(x_1)+\ldots+\alpha_l\phi(x_l)=X^T\alpha$$

donde α es un vector columna de dimensión $l\times 1$ que recoge los coeficientes de la combinación lineal, resulta que:

$$\frac{1}{l}\sum_{i=1}^{l}[v^T\phi(x_i)]^2=\frac{1}{l}v^T\left(X^T X\right)v=\frac{1}{l}\alpha^T(X X^T)(X X^T)\alpha$$

y como el elemento genérico de $X X^T$ es:

$$\left(X X^T\right)_{ij}=\phi(x_i)^T\phi(x_j)=\langle\phi(x_i),\phi(x_j)\rangle=k(x_i,x_j)=K_{ij}$$

resulta que:

$$X X^T=K$$

luego

$$\frac{1}{l}\sum_{i=1}^{l}[v^T\phi(x_i)]^2=\frac{1}{l}\alpha^T K^2\alpha$$

Es decir, bajo el supuesto de que los datos han sido centrados en el *feature space*, la formula anterior nos proporciona una medida de la variabilidad de las proyecciones de los elementos de $\phi(S)$ en la dirección de un vector unitario v obtenido como combinación lineal (con coeficientes recogidos en α) de los elementos de $\phi(S)$ y que depende exclusivamente de la matriz *kernel* K.

Tras presentar en este tema algunos algoritmos básicos que serán de utilidad más adelante y comprobar que es posible obtener mucha información del conjunto $\phi(S)$ contando exclusivamente con la información procedente de la matriz *kernel* K, dedicaremos los siguientes temas a presentar la aplicación de las funciones *kernel* al caso del aprendizaje supervisado y no supervisado.

4. Aplicación de las funciones *kernel* al aprendizaje supervisado: clasificadores de margen máximo y *support vector machines*

Una vez revisados en los temas anteriores los fundamentos teóricos de las funciones *kernel* y establecidos algunos algoritmos básicos que permiten obtener información a partir de la matriz *kernel* K, dedicaremos este tema a la aplicación de estos conceptos al caso del aprendizaje supervisado y, en concreto, a las llamadas *support vector machines* (o *máquinas de vector soporte* en su traducción al castellano).

Una *support vector machine*, como se detallará en este tema, es el resultado de combinar un *clasificador de margen máximo* con una *función kernel*. Así, el desarrollo de las *support vector machines* se ha apoyado en dos pilares:

- El desarrollo de los clasificadores de margen máximo, de la mano de Vapnik y Lerner (1963), Vapnik y Chervonenkis (1964), Cover (1965), Mangasarian (1965), Smith (1968), Duda y Hart (1973) o Bennett y Mangasarian (1992).

- El desarrollo de las funciones *kernel* y su aplicación en el ámbito del aprendizaje estadístico, debido a Mercer (1909), Aronszajn (1950), Aizerman, Braverman y Rozonoer (1964) o Poggio y Girosi (1990).

Finalmente, estas dos corrientes confluyen en los trabajos de Vapnik (1979, 1982), quien anuncia el nacimiento de las *support vector machines*; Boser, Guyon y Vapnik (1992), a quienes se debe la primera formulación de las *support vector machines* en su versión actual; Cortes y Vapnik (1995), quienes extendieron las *support vector machines* al caso del clasificador de margen máximo *soft* o al propio Vapnik (1995), quien extendió la aplicación de las *support vector machines* al caso de la regresión.

Las *support vector machines* constituyen un conjunto de métodos de aprendizaje supervisado que persigue la identificación de patrones y que tienen versiones tanto de clasificación (para variables cualitativas) como de regresión (para el caso de variables cuantitativas). Como una s*upport vector machine* es un clasificador (aunque un tanto sofisticado) su forma de funcionamiento habitual es la siguiente (consideraremos el caso más sencillo de una *support vector machine* de clasificación en dos clases):

- Se proporciona a la *support vector machine* un conjunto de elementos etiquetados, que constituyen lo que se da en llamar la muestra de aprendizaje o de entrenamiento (*training sample*).

- La *support vector machine* trata de determinar el *clasificador de margen máximo* (*hard* o *soft*, según veremos más adelante) entendiendo por tal aquel que maximiza la separación existente entre las dos clases de elementos. Esta forma de proceder busca minimizar el error de generalización del clasificador.

- A continuación se somete una nueva muestra de elementos etiquetados (denominada *muestra test*) al clasificador de margen máximo obtenido en el punto anterior. El propósito de este paso es el de evaluar de modo más objetivo la capacidad predictora del clasificador

obtenido.

La característica diferencial de las *support vector machines* reside en que, con carácter previo a la determinación del clasificador de margen máximo, los datos de la muestra son incrustados *(embedded)* en un *feature space* de dimensión muy elevada (incluso, infinita). El objetivo de esta transformación de los datos es el de facilitar su separabilidad ya que ésta es tanto más probable cuanto mayor sea la dimensión del espacio en el que se encuentran representados los datos.

Así, podemos decir que una *support vector machine* es una combinación (modular) de un clasificador de margen máximo (*hard* o *soft*) y de una función *kernel*, tal y como veremos en los siguientes apartados.

Comenzaremos por definir los clasificadores de margen máximo tanto en su versión *hard* como en su versión *soft* y, para cada uno de ellos, veremos dos formulaciones alternativas: la formulación primal y la formulación dual. Será esta última la que proporcionará la vía para la incorporación de la función *kernel* elegida al algoritmo de obtención del clasificador de margen máximo. En otras palabras, la formulación dual de los problemas de determinación de los clasificadores de margen máximo supone la *kernelización* de este procedimiento.

Clasificadores de margen máximo

Los *clasificadores de margen máximo* (*maximum-margin classifiers)* constituyen la base teórica sobre la que se construye el concepto de *support vector machine* que, como ya hemos comentado, es la combinación de un clasificador con una función *kernel.*

Nos centraremos en este apartado en los clasificadores de margen máximo en el caso de dos clases, que suelen estar etiquetadas como +1 y -1.

De una forma intuitiva (que un poco más adelante se concretará) se puede decir que el objetivo de un clasificador de margen máximo es encontrar una *superficie de decisión* (un hiperplano afín en el espacio en el que se encuentran representados los elementos a clasificar) que cumpla dos condiciones:

- que sea equidistante a las fronteras de las clases justo en aquel lugar en el que las clases se encuentran más próximas

- que, al mismo tiempo, maximice el *margen* existente entre las mencionadas clases.

Para que pueda establecerse el clasificador de margen máximo (al menos en su versión *hard*) es necesario que la muestra de aprendizaje sea una muestra *separable*.

Definición de muestra de aprendizaje separable en un problema de clasificación binaria: dada una muestra etiquetada

$$D = \{(x_1, y_1), \ldots, (x_l, y_l) \subset \mathbb{R}^n \times \{+1, -1\}\}$$

se dice que la muestra de aprendizaje es *separable* si las envolturas convexas de los elementos que conforman cada una de las clases tienen intersección vacía.

En este caso, la *superficie de decisión óptima* será equidistante de cada una de las dos clases. En cualquier caso, para definir con precisión el concepto de superficie de decisión óptima (cuya determinación es, precisamente, el objetivo nuclear de un clasificador de margen máximo) necesitamos establecer qué se entiende por *supporting hyperplane* y por *margen*.

Definición de *supporting hyperplane*: un hiperplano afín del espacio $\mathbb{R}^n$ es un *hiperplano soporte* (*supporting hyperplane)* de una clase si es paralelo a la superficie de decisión óptima y todos los puntos de la clase están bien por encima bien por debajo del hiperplano. Una forma de visualizar el hiperplano soporte es imaginándolo como una traslación de la superficie de decisión óptima hasta el punto en el que toca la frontera de su respectiva clase. En un problema de clasificación binario existe una superficie de decisión óptima y dos *supporting hyperplanes*.

Definición de margen: en un problema de clasificación binaria, llamamos *margen* a la distancia existente entre los dos *supporting hyperplanes*.

Ahora que ya hemos definido de manera precisa los conceptos de *supporting hyperplane* y de margen podemos establecer la definición de superficie de decisión óptima.

Definición: superficie de decisión óptima de un problema de clasificación binaria: una superficie de decisión de un problema de clasificación binaria es óptima si es equidistante de ambos *supporting hyperplanes* y maximiza su margen.

En resumen, el problema de determinación del clasificador de margen máximo en un problema separable consiste en localizar sendos hiperplanos afines en el espacio $\mathbb{R}^n$ (en el que se encuentran ubicados los elementos a clasificar) que deben:

- ser paralelos entre sí
- ser hiperplanos soporte de cada una de las dos clases respectivamente
- maximizar la distancia que existe entre ellos, es decir, el margen del clasificador

Una vez localizados estos dos hiperplanos soporte, la superficie óptima de decisión será el hiperplano afín de $\mathbb{R}^n$ equidistante de estos dos hiperplanos soporte.

Así, (considerando por el momento que los elementos a clasificar se encuentran ubicados en el plano $\mathbb{R}^2$), podemos visualizar el problema de encontrar la superficie de decisión óptima de un problema de clasificación binaria con muestra separable como la tarea de introducir un listón de máxima anchura entre las dos clases. La superficie de decisión óptima será, en este caso, la linea recta que atraviesa longitudinalmente el listón por la mitad.

Siguiendo con la analogía del listón, resulta evidente que algunos puntos de la muestra de aprendizaje no imponen ninguna restricción a la anchura del listón mientras que otros sí lo hacen. Naturalmente, los elementos que se encuentran más cercanos a los de la otra clase son los que suponen una mayor restricción. Aquellos puntos que determinan la anchura del listón reciben el nombre de *support vectors* (o vectores soporte).

Desde el punto de vista computacional conviene señalar que la tarea de encontrar el clasificador de margen máximo constituye un problema de *optimización convexa,* es decir, un problema en el que

la función objetivo es convexa y las restricciones son lineales y, en consecuencia, también convexas. Los problemas de optimización convexa gozan de la deseable propiedad de que las condiciones necesarias de óptimo son también condiciones suficientes, por lo que no cabe la existencia de óptimos globales alejados de óptimos locales, con las nefastas consecuencias que de cara al éxito del procedimiento esto podría suponer.

Determinación del clasificador de margen máximo:

Dada una muestra de aprendizaje separable:

$$D=\{(x_1, y_1),\ldots,(x_l, y_l)\subset\mathbb{R}^n\times\{+1,-1\}\}$$

la superficie de decisión de margen máximo se calcula resolviendo el siguiente problema de optimización:

$$min_{w,b}\frac{1}{2}w^T w$$

sujeto a las restricciones:

$$w^T(y_i x_i)\geq 1+y_i b \quad \forall (x_i, y_i)\in D$$

En efecto, el conjunto de soluciones factibles del problema de optimización lo constituyen todas las posibles superficies de decisión, de la forma $w^T x=b$ (siendo $w\in\mathbb{R}^n$ y b un escalar, por lo que $w^T x=b$ es un hiperplano afín en $\mathbb{R}^n$) con sus *supporting hyperplanes* asociados. Para cada una de estas posibles superficies de decisión debemos calcular qué margen permiten (es decir, cuál es la distancia entre sus *supporting hyperplanes*) y elegir aquella que haga máximo este margen. Así, la función objetivo será:

$$\phi(w, b)$$

que calcula el margen de una superficie de decisión determinada $w^T x=b$.

El óptimo (margen máximo) será:

$$m^*=\phi(w^*, b^*)=max_{w,b}\phi(w, b)$$

resultado de aplicar la función que calcula el margen al caso de la superficie de decisión óptima, que vendrá dada por $w^{*T} x=b^*$.

Vamos a determinar la función objetivo ϕ . Para ello vamos a considerar que disponemos de una muestra de aprendizaje etiquetada y separable:

$$D=\{(x_1, y_1),\ldots,(x_l, y_l)\subset\mathbb{R}^n\times\{+1,-1\}\}$$

y supongamos que hemos encontrado la superficie de decisión óptima $w^{*T} x=b^*$. Esta superficie debe cumplir que:

$$m^* = \phi(w^*, b^*) = max_{w,b}\phi(w, b)$$

Como esta superficie de decisión es óptima, tiene dos *supporting hyperplanes* equidistantes, es decir:

$$w^{*T}x = b^* + k \quad \text{y} \quad w^{*T}x = b^* - k$$

Ahora, por tratarse de los *supporting hyperplanes* de margen máximo, debe haber algún elemento de la clase +1 (llamémosle x_p) sobre el hiperplano $w^{*T}x = b^* + k$ y algún elemento de la clase -1 (llamémosle x_q) sobre el hiperplano $w^{*T}x = b^* - k$ es decir:

$$w^{*T}x_p = b^* + k \quad \text{y} \quad w^{*T}x_q = b^* - k$$

El margen es la distancia entre estos dos hiperplanos. Podemos calcularlo como el módulo de la proyección en la dirección de w^* del vector $x_p - x_q$ es decir:

$$m^* = \frac{w^{*T}(x_p - x_q)}{\|w^*\|} = \frac{w^{*T}x_p - w^{*T}x_q}{\|w^*\|} = \frac{(b^* + k) - (b^* - k)}{\|w^*\|} = \frac{2k}{\|w^*\|}$$

Así, si deseamos maximizar el margen, podemos maximizar $2k/\|w\|$ o, de modo equivalente, minimizar $\|w\|^2/(2k)$ ya que el valor de w que maximiza $2k/\|w\|$ es el mismo que minimiza $\|w\|/(2k)$ y, como $y = x^2$ es monótona creciente para $x > 0$, el valor de w que minimiza $\|w\|/(2k)$ es el mismo que minimiza $\|w\|^2/(2k)$

La ventaja de esta formulación de la función objetivo es que permite un tratamiento mediante programación cuadrática que, como veremos, simplifica el problema.

En resumen, la función objetivo (a minimizar) es:

$$\phi(w, b) = \frac{\|w\|^2}{2k}$$

El margen vendrá dado por:

$$\sqrt{\frac{2k}{\phi(w, b)}}$$

En cuanto a las restricciones, debemos tener en cuenta que, en todo momento, los *supporting hyperplanes* deben mantener correctamente clasificados a los elementos de cada clase. Es decir:

- para los elementos de la clase +1 debe ocurrir que $w^T x_i \geq b + k$
- para los de la clase -1 debe ocurrir que $w^T x_i \leq b - k$ o $w^T(-x_i) \geq k - b$

Teniendo en cuenta la etiqueta de las clases podemos escribir:

$$w^T(y_i x_i) \geq b\, y_i + k \qquad \forall (x_i, y_i) \in D$$

Por último, elegimos un valor arbitrario para la constante k (que aparece tanto en la función objetivo como en las restricciones). Si hacemos $k=1$ obtenemos que la función objetivo es:

$$\phi(w,b)=\frac{\|w\|^2}{2}$$

Por su parte las restricciones son:

$$w^T(y_i x_i)\geq 1+y_i b \quad \forall (x_i,y_i)\in D$$

Y el margen se obtiene haciendo:

$$\sqrt{\frac{2}{\phi(w,b)}}=\frac{2}{\|w\|}$$

Formulación dual del clasificador de margen máximo

Una vez definido el problema de optimización que permite la determinación de la superficie óptima para la clasificación de dos clases separables, procederemos a continuación a encontrar una formulación alternativa del problema, que recibe el nombre de *formulación dual*. Esto hace que la formulación original (la que hemos tratado en el apartado anterior) se denomine *formulación primal*.

Para encontrar la formulación dual del problema de determinación del clasificador de margen máximo precisamos definir algunos conceptos previos.

Definición: Problema de optimización lagrangiano

Dado el problema de optimización siguiente (llamado problema *primal*):

$$min_x \phi(x)$$

sujeto a :

$$g_i(x)\geq 0 \qquad \forall i\in\{1,\ldots,l\}$$

donde $\phi:\mathbb{R}^n\rightarrow\mathbb{R}$ es una función objetivo convexa y g_i son restricciones lineales $\mathbb{R}^n\rightarrow\mathbb{R}$ y, en consecuencia, convexas, se llama *problema de optimización lagrangiano* a:

$$max_\alpha min_x L(\alpha,x)=max_\alpha min_x\left(\phi(x)-\sum_{i=1}^{l}\alpha_i g_i(x)\right)$$

sujeto a:

$$\alpha_i\geq 0 \qquad \forall i\in\{1,\ldots,l\}$$

A la función

$$L(\alpha, x) = \left(\phi(x) - \sum_{i=1}^{l} \alpha_i g_i(x) \right)$$

se le llama *función lagrangiana* .

A los valores

$$\alpha = (\alpha_1, \ldots, \alpha_l)$$

se les denomina *multiplicadores lagrangianos* y a $\alpha \in \mathbb{R}^l$, *variable dual*

Las soluciones a $max_\alpha min_x L(\alpha, x)$ son los puntos silla (*saddle points*) en el gráfico de la función $L(\alpha, x)$ pero, como la función objetivo es convexa y las restricciones son lineales, $L(\alpha, x)$ tiene un único punto silla. En dicho punto silla la derivada parcial de $L(\alpha, x)$ respecto a x debe anularse, es decir:

$$\frac{dL}{dx} = 0$$

si llamamos x^* al valor de x en el punto silla, resultará que:

$$\frac{dL}{dx}(\alpha, x^*) = 0$$

Bajo ciertas condiciones, una solución del problema de optimización lagrangiano es también una solución del problema de optimización primal.

En efecto, si (α^*, x^*) es una solución del problema de optimización lagrangiano, es decir, si:

$$max_\alpha min_x L(\alpha, x) = L(\alpha^*, x^*) = \left(\phi(x^*) - \sum_{i=1}^{l} \alpha_i^* g_i(x^*) \right)$$

entonces x^* es una solución del problema de optimización primal si y sólo si:

1. $\frac{dL}{dx}(\alpha, x^*) = 0$
2. $\alpha_i^* g_i(x^*) = 0 \quad \forall i \in \{1, \ldots, l\}$
3. $g_i(x^*) \geq 0 \quad \forall i \in \{1, \ldots, l\}$
4. $\alpha_i^* \geq 0 \quad \forall i \in \{1, \ldots, l\}$

Estas cuatro condiciones se llaman, colectivamente, *condiciones de Karush-Kuhn-Tucker* (o condiciones KKT). En particular, la segunda recibe el nombre de *condición de complementariedad de Karush-Kuhn-Tucker.*

En el caso de que la función objetivo sea convexa, el óptimo del problema de optimización primal debe estar necesariamente en el único punto silla de la función lagrangiana. En consecuencia, a partir de la ecuación

$$\frac{dL}{dx}(\alpha, x^*)=0$$

es posible reformular el problema de optimización lagrangiano en función, exclusivamente, de la variable dual, es decir,

$$L(\alpha, x^*)=\phi'(\alpha)$$

Así, el problema de optimización será:

$$max_\alpha \phi'(\alpha)$$

sujeto a

$$\alpha_i \geq 0 \qquad \forall i \in \{1,\ldots,l\}$$

que recibe el nombre de *problema de optimización dual* (o *dual lagrangiano* o *dual de Wolfe*).

En resumen, este camino permite resolver el problema de optimización primal usando el problema dual ya que:

$$max_\alpha \phi'(\alpha)=\phi'(\alpha^*)=L(\alpha^*, x^*)=\phi(x^*)$$

Además, podemos estar seguros de que en el óptimo se cumplen las condiciones de KKT.

Después de esta presentación de los conceptos de problema de optimización lagrangiano y problema dual, estamos ya en condiciones de obtener la *expresión dual del problema del clasificador de margen máximo*.

Recordemos que el problema de optimización (primal) del clasificador de margen máximo es:

Minimizar:

$$\phi(w, b)=\frac{\|w\|^2}{2}$$

sujeto a

$$w^T(y_i x_i) \geq 1+y_i b \quad \forall (x_i, y_i) \in D$$

Podemos reformular las restricciones para que permitan la construcción de la función lagrangiana:

$$y_i(w^T x_i - b)-1 \geq 0 \quad \forall (x_i, y_i) \in D$$

La función lagrangiana es:

$$L(\alpha, w, b) = \frac{1}{2} w^T w - \sum_{i=1}^{l} \alpha_i (y_i (w^T x_i - b) - 1)$$

y operando de modo elemental:

$$L(\alpha, w, b) = \frac{1}{2} w^T w - \sum_{i=1}^{l} \alpha_i y_i w^T x_i + b \sum_{i=1}^{l} \alpha_i y_i + \sum_{i=1}^{l} \alpha_i$$

En consecuencia, el problema de optimización lagrangiano asociado al problema de optimización primal del clasificador de margen máximo es:

$$max_{\alpha} min_{w,b} L(\alpha, w, b)$$

sujeto a:

$$\alpha_i \geq 0 \qquad \forall i \in \{1, \ldots, l\}$$

Supongamos que (α^*, w^*, b^*) es una solución del problema de optimización lagrangiano, es decir, supongamos que:

$$L(\alpha^*, w^*, b^*) = max_{\alpha} min_{w,b} L(\alpha, w, b)$$

Entonces, como la función objetivo del problema primal es convexa y las restricciones del problema primal son lineales, podemos asegurar que la solución (α^*, w^*, b^*) del problema de optimización lagrangiano asociado satisfará las siguientes condiciones (condiciones de KKT):

1. $\frac{\partial L}{\partial w}(\alpha, w^*, b) = 0$

2. $\frac{\partial L}{\partial b}(\alpha, w, b^*) = 0$

3. $\alpha_i^* (y_i (w^{*T} x_i - b^*) - 1) = 0 \qquad \forall i \in \{1, \ldots, l\}$

4. $y_i (w^{*T} x_i - b^*) - 1 \geq 0 \qquad \forall i \in \{1, \ldots, l\}$

5. $\alpha_i^* \geq 0 \qquad \forall i \in \{1, \ldots, l\}$

El significado de estas condiciones es el siguiente:

- Las dos primeras condiciones aseguran que la solución (α^*, w^*, b^*) constituye el único punto silla de

$$L(\alpha, w, b) = \frac{1}{2} w^T w - \sum_{i=1}^{l} \alpha_i y_i w^T x_i + b \sum_{i=1}^{l} \alpha_i y_i + \sum_{i=1}^{l} \alpha_i$$

- La tercera condición es la llamada condición de complementariedad y garantiza que la

solución del problema de optimización lagrangiano coincide con la solución del problema primal.

- Las dos últimas condiciones recogen las restricciones planteadas en el problema primal y en el problema de optimización lagrangiano respectivamente.

Dadas las particularidades de la función objetivo (convexa) y de las restricciones (lineales) del problema de optimización primal, podemos construir el problema de optimización dual (dual lagrangiano).

Para ello, a partir de la primera condición de KKT, calculamos la derivada parcial de la función lagrangiana con respecto a la variable primal w, la evaluamos en la solución w^* y la igualamos a cero.

$$\frac{\partial L}{\partial w}(\alpha, w^*, b) = w^* - \sum_{i=1}^{l} \alpha_i y_i x_i = 0$$

de donde se obtiene que

$$w^* = \sum_{i=1}^{l} \alpha_i y_i x_i$$

Ahora, aplicando la segunda condición de KKT y procediendo de modo análogo tenemos que:

$$\frac{\partial L}{\partial b}(\alpha, w, b^*) = \sum_{i=1}^{l} \alpha_i y_i = 0$$

Si sustituimos estas dos últimas expresiones en la función lagrangiana tendremos:

$$L(\alpha, w^*, b^*) = \frac{1}{2}\left[\sum_{i=1}^{l} \alpha_i y_i x_i\right]^T \left[\sum_{i=1}^{l} \alpha_i y_i x_i\right] - \sum_{i=1}^{l} \alpha_i y_i \left[\sum_{i=1}^{l} \alpha_i y_i x_i\right]^T x_i + b \times 0 + \sum_{i=1}^{l} \alpha_i$$

es decir:

$$L(\alpha, w^*, b^*) = \sum_{i=1}^{l} \alpha_i + \frac{1}{2}\left[\sum_{i=1}^{l} \alpha_i y_i x_i\right]^T \left[\sum_{i=1}^{l} \alpha_i y_i x_i\right] - \sum_{i=1}^{l} \alpha_i y_i \left[\sum_{i=1}^{l} \alpha_i y_i x_i\right]^T x_i$$

y, en definitiva:

$$L(\alpha, w^*, b^*) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

que no depende de w ni de b. Hemos obtenido así el problema de optimización dual asociado al clasificador de margen máximo siguiente:

Maximizar:

$$\phi'(\alpha)=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

sujeto a las restricciones

$$\sum_{i=1}^{l}\alpha_i y_i=0$$

$$\alpha_i\geq 0 \quad \forall i\in\{1,\ldots,l\}$$

A la luz del problema de optimización dual es sencillo interpretar la condición de complementariedad de KKT, que recordemos, afirmaba que debe verificarse:

$$\alpha_i^*(y_i(w^{*T}x_i-b^*)-1)=0 \quad \forall i\in\{1,\ldots,l\}$$

Sabemos, por otra parte, que también debe cumplirse que:

$$y_i(w^{*T}x_i-b^*)-1\geq 0 \quad \forall i\in\{1,\ldots,l\}$$

En consecuencia para que se cumpla la condición de complementariedad de KKT debe ocurrir que:

- si $\alpha_i^*>0$ entonces $y_i(w^{*T}x_i-b^*)-1=0$. En otras palabras, aquellos elementos de la muestra de aprendizaje para los que el valor del multiplicador α_i^* sea positivo, deben cumplir que se encuentran sobre su correspondiente *supporting hyperplane*. Son los llamados vectores soporte (*support vectors*)

- si $y_i(w^{*T}x_i-b^*)-1>0$ entonces $\alpha_i^*=0$. En otras palabras, aquellos elementos de la muestra de aprendizaje que no se encuentran sobre su *supporting hyperplane*, tienen un valor nulo en su correspondiente multiplicador.

Si ahora recordamos que:

$$w^*=\sum_{i=1}^{l}\alpha_i^* y_i x_i$$

comprobamos que son únicamente los vectores soporte (elementos de la muestra de aprendizaje con $\alpha_i^*>0$) los que determinan la posición de la frontera de decisión óptima en el problema del clasificador de margen máximo.

Finalmente, para calcular el valor de b^* podemos proceder seleccionando un *support vector* x_p . Como el *support vector* debe estar sobre su correspondiente *supporting hyperplane* debe cumplirse que:

$$y_p(w^{*T}x_p-b^*)-1=0$$

y, teniendo en cuenta que $y_i \in \{+1,-1\} \ \forall i \in \{1,\ldots,l\}$, resulta que:

$$w^{*T} x_p - b^* = y_p$$

y, por tanto:

$$b^* = w^{*T} x_p - y_p$$

Sustituyendo el valor de w^* :

$$b^* = \sum_{i=1}^{l} \alpha_i^* y_i x_i^T x_p - y_p$$

Como resumen de todo lo anterior podemos tratar de interpretar los distintos enfoques primal-dual frente al problema de determinación del clasificador de margen máximo. Podemos decir que:

- El problema de optimización primal se enfrenta al problema de determinación del clasificador óptimo buscando en el espacio de todas las posibles superficies de separación de las dos clases aquélla que permite un margen máximo. El resultado del problema de optimización es un hiperplano afín y sólo a partir de ese hiperplano podemos establecer cuáles son los *supporting hyperplanes* y, por tanto, cuáles son los vectores soporte: los que se encuentran sobre su correspondiente *supporting hyperplane*.

- El problema de optimización dual se enfrenta al problema buscando en el conjunto

  $$\left[\mathbb{R}^+ \cup \{0\}\right]^l$$

  una asignación de pesos (multiplicadores) a los diferentes elementos que componen la muestra de aprendizaje. Estos multiplicadores (que deben ser no negativos y cuya suma en ambas clases debe ser la misma) establecen cuáles son los elementos de la muestra de aprendizaje que determinan la posición de la superficie óptima de separación. Así, habrá elementos de la muestra de aprendizaje cuyo multiplicador es nulo y que, en consecuencia, no tendrán ninguna influencia en la determinación de la superficie de decisión óptima. Otros, sin embargo, tendrán asignado un multiplicador positivo y, por tanto, serán los que determinarán la posición de la superficie de decisión óptima. Son estos últimos, los llamados vectores soporte (*support vectors*), el resultado básico del problema dual. En vez de determinar la función de separación óptima (como hace el problema primal), el problema dual establece cuáles son los vectores soporte. A partir de ahí, el cálculo de la superficie óptima de decisión, es inmediato.

La formulación dual del problema de clasificación de margen máximo no ha sido un mero ejercicio intelectual. Como se verá un poco más adelante, es precisamente esta formulación dual del problema de optimización la que abre la puerta a la incorporación de las funciones *kernel* y, en consecuencia, a la aplicación de las *support vector machines*. No obstante, antes de entrar en el asunto de la *kernelización*, permitamos, en el siguiente apartado, que el clasificador de margen máximo cometa algunos errores. Aunque a primera vista esto parezca una desventaja, veremos que la idea no es en absoluto disparatada.

Clasificadores de margen flexible (soft-margin classifiers)

Como ya se ha señalado en reiteradas ocasiones, la separabilidad de las dos clases analizadas es condición necesaria para la determinación del clasificador de margen máximo. En otras palabras, los clasificadores de margen máximo, tal como los hemos definido hasta el momento, funcionan bajo la condición de que la intersección de las envolturas convexas de las clases sea vacía.

En las aplicaciones reales, no obstante, lo más común es que las clases consideradas no sean separables, lo que hace inaplicable el clasificador de margen máximo analizado en el apartado anterior. Ante esta situación caben distintas vías de actuación:

- Una posible vía para solventar este problema es incrustar los elementos de la muestra de aprendizaje en un espacio de dimensión mayor (quizás infinita) en el que sí puedan ser linealmente separables. Las funciones *kernel*, presentadas en anteriores apartados, son adecuadas para este fin, ya que permiten operar en el espacio de mayor dimensión sin tener que hacer frente a las dificultades computacionales que esta mayor dimensión acarrearía.

- Otra posible vía para solventar la inaplicabilidad del clasificador de margen máximo al caso de clases no perfectamente separables es la de permitir que el clasificador "cometa algunos errores" y no tome en consideración, a la hora de determinar la frontera óptima de decisión, algunas de las observaciones que pueden ser consideradas como "ruido estadístico".

A pesar de que la idea de permitir al clasificador cometer algunos errores puede parecer inadecuada, la ventaja de esta forma de proceder es que lleva a superficies de separación óptima más simples y, por tanto, con una mayor generalizabilidad y menor riesgo de *overfitting*. Por este motivo, exploraremos, a continuación, los clasificadores de margen máximo a los que (valga la expresión) "se les permite cometer algunos errores". En la literatura reciben el nombre de *soft-margin classifiers*, en oposición a los *hard-margin classifiers,* que son los que han sido presentados en el apartado anterior. Esto no significa que renunciemos a la posibilidad de recurrir a las funciones *kernel* para incrustar los datos en un espacio de dimensión mayor. Como se verá más adelante, la fórmula que proporciona mejores resultados es una combinación de estas dos vías, es decir, permitir que el clasificador cometa algunos errores cuando se le pide que clasifique a los elementos en un espacio de dimensión elevada definido por la elección de determinada función *kernel*.

Recordemos la formulación primal del problema de determinación del clasificador de margen máximo para dos clases separables:

Clasificador de margen máximo: Dada una muestra de aprendizaje separable

$$D = \{(x_1, y_1), \ldots, (x_l, y_l) \subset \mathbb{R}^n \times \{+1, -1\}\}$$

la superficie de decisión de margen máximo se calcula resolviendo el siguiente problema de optimización:

$$min_{w,b} \frac{1}{2} w^T w$$

sujeto a las restricciones:

$$y_i(w^T x_i - b) - 1 \geq 0 \quad \forall (x_i, y_i) \in D$$

Observando la formulación de este problema es fácil comprobar que la ubicación de la superficie óptima puede verse muy afectada (y, en consecuencia, ser poco robusta) por la presencia de unos pocos elementos en la muestra de aprendizaje que pueden no ser significativos. Para evitar que esto suceda, puede procederse introduciendo unas variables de holgura no negativas (*slack variables*) cuyo propósito es permitir que algunos de los puntos de la muestra de aprendizaje se encuentren en el lado incorrecto de su *supporting hyperplane*. Con este fin, las restricciones se transforman en:

$$y_i(w^T x_i - b) + \xi_i - 1 \geq 0 \quad \forall (x_i, y_i) \in D$$

$$\xi_i \geq 0 \qquad \forall i \in \{1, \ldots, l\}$$

Para evitar que el problema de optimización tenga como solución una que resulte trivial (haciendo que todas las variables de holgura tomen valores elevados y que, en consecuencia, las observaciones no influyan en la determinación de la superficie óptima de decisión), es necesario incluir dichas variables de holgura en la función objetivo, en forma de penalización. Así, la función objetivo se convierte en:

$$\frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \quad \text{con} \quad C > 0$$

En resumen:

Clasificador de margen máximo soft (soft-margin classifier): Dada una muestra de aprendizaje (no necesariamente separable):

$$D = \{(x_1, y_1), \ldots, (x_l, y_l) \subset \mathbb{R}^n \times \{+1, -1\}\}$$

el problema de optimización primal asociado a la determinación del clasificador de margen máximo *soft* (*soft-margin classifier*) es:

$$min_{w, \xi, b} \left(\frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \right) \quad \text{con} \quad C > 0$$

sujeto a las restricciones:

$$y_i(w^T x_i - b) + \xi_i - 1 \geq 0 \qquad \forall i \in \{1, \ldots, l\}$$

$$\xi_i \geq 0 \qquad \forall i \in \{1, \ldots, l\}$$

En el caso del clasificador de margen máximo *soft*, se debe encontrar un compromiso entre el tamaño del margen y el número de elementos que no se encuentran en el lado correcto de su *supporting hyperplane*. Naturalmente, cuanto menor sea el margen, menor será el número de errores y viceversa. La constante C, que aparece en la función objetivo es el llamado *parámetro de coste* y permite controlar este *trade-off*:

- Si el valor de C es muy alto, se está indicando que el coste de cometer errores es muy

elevado. La solución óptima del problema asignará valores muy bajos a las variables de holgura, es decir, nos llevará a un margen muy estrecho y a un reducido número de elementos mal clasificados.

- Por el contrario, un valor de C muy bajo está indicando que el coste de los errores es pequeño, lo que permitirá que las variables de holgura tomen valores más elevados. En consecuencia, un valor reducido de C llevará a márgenes mayores con una mayor proporción de elementos mal clasificados.

Es importante interpretar correctamente el significado que tiene el valor de las variables de holgura en la solución del problema primal:

- Si para un elemento determinado de la muestra resulta que $\xi_i = 0$, esto significa que para esa observación no ha sido necesario asignar ningún valor a la variable de holgura para que "parezca" que se encuentra en el lado correcto del correspondiente *supporting hyperplane*. En consecuencia, la observación se encuentra en el lado correcto de su *supporting hyperplane*.

- Si para un elemento de la muestra resulta que $0 < \xi_i \leq 1$, ha sido necesario asignar un valor a la variable de holgura para que parezca que la observación se encuentra en el lado correcto del *supporting hyperplane*. En consecuencia, el elemento se encuentra en el lado incorrecto de su *supporting hyperplane*. Sin embargo, se encuentra en el lado correcto de la superficie óptima de decisión. Para entender esto es necesario recordar las ecuaciones de los *supporting hyperplanes*. Vamos a considerar separadamente los elementos de cada una de las dos clases:

  - Si $y_i = +1$, entonces la restricción se convierte en $(w^T x_i - b) \geq 1 - \xi_i \geq 0$, es decir, $w^T x_i \geq b$, y el elemento se encuentra en el lado correcto de la superficie óptima (aunque en el lado incorrecto de su *supporting hyperplane*).

  - Si $y_i = -1$, entonces la restricción se convierte en $(w^T x_i - b) \leq \xi_i - 1 \leq 0$, es decir, $w^T x_i \leq b$, y el elemento se encuentra en el lado correcto de la superficie óptima (aunque en el lado incorrecto de su *supporting hyperplane*).

- Por último, si para algún elemento de la muestra de aprendizaje resulta que el valor de la variable de holgura en la solución es $\xi_i > 1$, entonces el elemento se encuentra no sólo en el lado incorrecto de su *supporting hyperplane*, sino también en el lado incorrecto de la superficie óptima. En efecto:

  - Si $y_i = +1$, entonces para que la restricción $(w^T x_i - b) + \xi_i - 1 \geq 0$ alcance el valor cero ha sido necesario que $\xi_i > 1$ por lo que $w^T x_i < b$ y el elemento se encuentra en el lado incorrecto de la superficie óptima.

  - Si $y_i = -1$, entonces para que la restricción $-(w^T x_i - b) + \xi_i - 1 \geq 0$ alcance el valor cero ha sido necesario que $\xi_i > 1$ por lo que $-(w^T x_i - b) < 0$ y, en consecuencia, $w^T x_i > b$, lo que significa que la observación se encuentra en la lado incorrecto de la superficie de decisión óptima.

En resumen:

- Si $\xi_i = 0$ entonces el elemento está en el lado correcto del *supporting hyperplane*
- Si $0 < \xi_i \leq 1$ entonces el elemento está en el lado incorrecto del *supporting hyperplane* pero en el lado correcto de la superficie óptima de decisión
- Si $\xi_i > 1$ entonces el elemento está en el lado incorrecto de la superficie óptima de decisión (y naturalmente, en el lado incorrecto de su *supporting hyperplane*)

Una vez formulado el problema de optimización primal asociado al clasificador de margen máximo *soft* estableceremos a continuación el problema de optimización dual que, recordemos, proporcionará la vía para la incorporación de las funciones *kernel* a los clasificadores.

Para ello comenzamos por la construcción de la función lagrangiana:

$$L(\alpha, \beta, w, \xi, b) = \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i - \sum_{i=1}^{l} \alpha_i (y_i (w^T x_i - b) + \xi_i - 1) - \sum_{i=1}^{l} \beta_i \xi_i$$

donde w , ξ y b son las variables primales y α y β las variables duales

El problema de optimización lagrangiano asociado al problema de optimización primal del clasificador de margen máximo *soft* es:

$$max_{\alpha, \beta} min_{w, \xi, b} L(\alpha, \beta, w, \xi, b)$$

sujeto a las restricciones:

$$\alpha_i \geq 0 \quad \forall i \in \{1, \ldots, l\}$$

$$\beta_i \geq 0 \quad \forall i \in \{1, \ldots, l\}$$

Al igual que en el caso *hard*, el hecho de que la función objetivo sea convexa unido al carácter lineal de las restricciones hace que la función lagrangiana presente un único punto silla y, en consecuencia, la solución del problema de optimización lagrangiano $\alpha^*, \beta^*, w^*, \xi^*, b^*$ debe satisfacer las siguientes condiciones de KKT:

1. $\frac{\partial L}{\partial w}(\alpha, \beta, w^*, \xi, b) = 0$
2. $\frac{\partial L}{\partial \xi_i}(\alpha, \beta, w, \xi_i^*, b) = 0 \quad \forall i \in \{1, \ldots, l\}$
3. $\frac{\partial L}{\partial b}(\alpha, \beta, w, \xi, b^*) = 0$
4. $\alpha_i^* (y_i (w^{*T} x_i - b^*) + \xi_i^* - 1) = 0 \quad \forall i \in \{1, \ldots, l\}$

5. $\beta_i^* \xi_i^* = 0 \quad \forall i \in \{1,\ldots,l\}$

6. $y_i(w^{*T} x_i - b^*) + \xi_i^* - 1 \geq 0 \quad \forall i \in \{1,\ldots,l\}$

7. $\alpha_i^* \geq 0 \quad \forall i \in \{1,\ldots,l\}$

8. $\beta_i^* \geq 0 \quad \forall i \in \{1,\ldots,l\}$

9. $\xi_i^* \geq 0 \quad \forall i \in \{1,\ldots,l\}$

Este conjunto de condiciones garantiza que podemos emplear el problema de optimización lagrangiano para resolver el problema de optimización primal. Es más, podemos aprovechar la estructura convexa de la función objetivo primal y la estructura lineal de las restricciones de dicho problema de optimización para construir el problema dual asociado.

Para ello derivaremos la función lagrangiana respecto a las variables primales, evaluaremos tales derivadas en el punto silla, las igualaremos a cero y sustituiremos los resultados en la función lagrangiana.

Derivando la función lagrangiana con respecto a w, evaluándola en w^*, e igualándola a cero obtenemos:

$$\frac{\partial L}{\partial w}(\alpha, \beta, w^*, \xi, b) = w^* - \sum_{i=1}^{l} \alpha_i y_i x_i = 0$$

y, por tanto:

$$w^* = \sum_{i=1}^{l} \alpha_i y_i x_i$$

Procediendo de manera análoga con respecto a b tenemos que:

$$\frac{\partial L}{\partial b}(\alpha, \beta, w, \xi, b^*) = \sum_{i=1}^{l} \alpha_i y_i = 0$$

Finalmente, procediendo análogamente con las variables de holgura obtenemos que:

$$\frac{\partial L}{\partial \xi_i}(\alpha, \beta, w, \xi_i^*, b) = C - \alpha_i - \beta_i = 0 \quad \forall i \in \{1,\ldots,l\}$$

Ahora, tomamos la función lagrangiana y la adaptamos para poder sustituir fácilmente el valor de w^* y tomar en cuenta las restricciones.

Recordemos que la función objetivo era:

$$L(\alpha, \beta, w, \xi, b) = \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i - \sum_{i=1}^{l} \alpha_i (y_i (w^T x_i - b) + \xi_i - 1) - \sum_{i=1}^{l} \beta_i \xi_i$$

Operando con los sumatorios y reordenando:

$$L(\alpha,\beta,w,\xi,b)=\frac{1}{2}w^T w+\sum_{i=1}^{l}\left[(C-\alpha_i-\beta_i)\xi_i-\alpha_i y_i w^T x_i+\alpha_i y_i b+\alpha_i\right]$$

Como $w^T x_i=x_i^T w$

$$L(\alpha,\beta,w,\xi,b)=\frac{1}{2}w^T w+\sum_{i=1}^{l}\left[(C-\alpha_i-\beta_i)\xi_i-\alpha_i y_i x_i^T w+\alpha_i y_i b+\alpha_i\right]$$

con lo que:

$$L(\alpha,\beta,w,\xi,b)=\frac{1}{2}w^T w+\sum_{i=1}^{l}\left[(C-\alpha_i-\beta_i)\xi_i\right]-\sum_{i=1}^{l}\left[\alpha_i y_i x_i^T\right]w+b\sum_{i=1}^{l}\alpha_i y_i+\sum_{i=1}^{l}\alpha_i$$

Es decir,

$$L(\alpha,\beta,w^*,\xi^*,b^*)=\frac{1}{2}w^{*T} w^*+\sum_{i=1}^{l}\left[(C-\alpha_i-\beta_i)\xi_i^*\right]-\sum_{i=1}^{l}\left[\alpha_i y_i x_i^T\right]w^*+b^*\sum_{i=1}^{l}\alpha_i y_i+\sum_{i=1}^{l}\alpha_i$$

Como

$$w^*=\sum_{i=1}^{l}\alpha_i y_i x_i \quad ,$$

$$C-\alpha_i-\beta_i=0 \qquad \forall i\in\{1,\ldots,l\} \quad \text{y}$$

$$\sum_{i=1}^{l}\alpha_i y_i=0$$

la función lagrangiana se transforma en:

$$L(\alpha,\beta,w^*,\xi^*,b^*)=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}w^{*T} w^*$$

Finalmente, sustituyendo el valor de w^* tenemos el dual lagrangiano siguiente:

$$\phi'(\alpha)=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j x_i^T x_j$$

que tiene la misma forma que la función objetivo del problema de optimización dual asociado al clasificador de margen máximo *hard*.

En cuanto a las restricciones del problema dual debemos tener en cuenta que como:

- $\alpha_i\geq 0 \qquad \forall i\in\{1,\ldots,l\}$

- $\beta_i \geq 0 \quad \forall i \in \{1,\ldots,l\}$
- $C - \alpha_i - \beta_i = 0 \quad \forall i \in \{1,\ldots,l\}$

debe ocurrir que $0 \leq \alpha_i \leq C \quad \forall i \in \{1,\ldots,l\}$

En definitiva, el problema dual asociado al problema del clasificador de margen máximo *soft* viene dado por:

Maximizar:

$$\phi'(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

sujeto a:

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C \quad \forall i \in \{1,\ldots,l\}$$

Este problema tiene como única variable (vectorial) α, cuyos elementos tienen valores que están acotados entre 0 y C (ambos incluidos). Veamos el significado de los valores de α en la solución:

- Si $\alpha_i^* = 0$ entonces, necesariamente $\beta_i^* = C$, con lo que por la quinta condición de KKT resulta que $\xi_i^* = 0$. En consecuencia, el elemento se encuentra correctamente situado en su *supporting hyperplane* y por ser $\alpha_i^* = 0$ no participa en la determinación de la superficie óptima.

- Si $0 < \alpha_i^* < C$ entonces $0 < \beta_i^* < C$, con lo que por la quinta condición de KKT resulta que $\xi_i^* = 0$. En consecuencia, el elemento se encuentra correctamente situado respecto a su *supporting hyperplane* y por ser $0 < \alpha_i^* < C$ participa en la determinación de la superficie óptima. Los elementos que cumplen $0 < \alpha_i^* < C$ son los *support vectors*.

- Si $\alpha_i^* = C$ entonces necesariamente $\beta_i^* = 0$, con lo que la quinta condición de KKT se satisface aun cuando $\xi_i^* > 0$. En este caso, el elemento se encontraría incorrectamente clasificado por su *supporting hyperplane*.

Una vez establecido el valor de la variable α en el problema dual, el calculo de w^* se realiza fácilmente a través de la siguiente fórmula:

$$w^* = \sum_{i=1}^{l} \alpha_i^* y_i x_i$$

Finalmente, para determinar el valor de b^* basta con tener en cuenta que cualquiera de los

support vectors se encuentra situado en su correspondiente *supporting hyperplane* por lo que debe cumplir que

$$y_p(w^{*T}x_p - b^*) - 1 = 0$$

y, teniendo en cuenta que $y_i \in \{+1,-1\} \ \forall i \in \{1,\ldots,l\}$, resulta que:

$$b^* = w^{*T}x_p - y_p$$

Sustituyendo el valor de w^* :

$$b^* = \sum_{i=1}^{l} \alpha_i^* y_i x_i^T x_p - y_p$$

En este punto hay que tener especial cuidado en la definición de *support vector* ya que, a diferencia de lo que ocurre en el caso del *hard-margin classifier*, un valor de $\alpha_i^* = C$ significa que el elemento está mal ubicado en relación con su *supporting hyperplane*. En consecuencia, para la determinación del valor de b^* es necesario seleccionar un elemento de la muestra para el que $0 < \alpha_i^* < C$, ya que esta condición garantiza que el elemento se encuentra situado justo en su *supporting hyperplane*.

Una vez presentadas las formulaciones duales para los problemas de determinación de los clasificadores de margen máximo (en sus versiones tanto *hard* como *soft),* estamos en condiciones de incorporar las funciones *kernel* (y, por tanto, de describir las *support vector machines*). En efecto, si prestamos atención a las funciones objetivo de los problemas duales, vemos que los valores de los elementos a clasificar aparecen únicamente a través de los productos escalares entre ellos. Esto nos permite afirmar que la formulación dual del problema de determinación de los clasificadores de margen máximo constituye la *kernelización* del procedimiento. Dedicamos la próxima sección a estudiarlo con más detalle.

Clasificadores de margen máximo y funciones kernel

La formulación dual del problema de optimización asociado al clasificador de margen máximo (tanto en su versión *hard* como en su versión *soft*) abre la puerta a la aplicación de las funciones *kernel* a estas tareas de aprendizaje estadístico.

En efecto, en la formulación dual de estos problemas, los valores de los elementos que componen la muestra de aprendizaje aparecen únicamente expresados en función de sus productos escalares. Dicho de otra forma, la formulación dual de los problemas de clasificación mediante margen máximo (tanto *hard* como *soft*) supone la *kernelización* de este problema de aprendizaje estadístico.

En consecuencia, sustituyendo los productos escalares $x_i^T x_j$ por una función *kernel* $k(x_i, x_j)$ genérica tendremos que el problema del *clasificador kernel de margen máximo* se puede resolver a través del siguiente problema de optimización:

Maximizar:

$$\phi'(\alpha)=\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\alpha_i\alpha_j y_i y_j k(x_i,x_j)$$

sujeto a:

$$\sum_{i=1}^{l}\alpha_i y_i=0$$

$$0\leq\alpha_i\leq C \quad \forall i\in\{1,\ldots,l\}$$ (en el caso del *soft-margin classifier*)

$$\alpha_i\geq 0 \quad \forall i\in\{1,\ldots,l\}$$ (en el caso del *hard-margin classifier*)

El valor de w^* resultará ser:

$$w^*=\sum_{i=1}^{l}\alpha_i^* y_i\phi(x_i)$$

Es interesante destacar que, como habitualmente la función de *embedding* $\phi: X\to F$ será desconocida, no será posible conocer el valor de w^*. No obstante, como veremos a continuación, esto no supone ningún problema.

En cuanto al valor de b^*:

$$b^*=\sum_{i=1}^{l}\alpha_i^* y_i\phi(x_i)^T\phi(x_p)-y_p=\sum_{i=1}^{l}\alpha_i^* y_i k(x_i,x_p)-y_p$$

con lo que la frontera óptima de decisión tendrá por ecuación:

$$w^{*T}\phi(x)=b^*$$

es decir:

$$\sum_{i=1}^{l}\alpha_i^* y_i\phi(x_i)^T\phi(x)=\sum_{i=1}^{l}\alpha_i^* y_i k(x_i,x_p)-y_p$$

y, en definitiva:

$$\sum_{i=1}^{l}\alpha_i^* y_i k(x_i,x)=\sum_{i=1}^{l}\alpha_i^* y_i k(x_i,x_p)-y_p$$

En resumen, a pesar de que si desconocemos la función de *embedding* ϕ resultará imposible conocer w^*, esto no supone ningún problema para determinar la ecuación de la frontera óptima de decisión que, como es evidente, será un hiperplano afín en el *feature space* pero tendrá forma potencialmente no lineal en el *input space* (en el que será una variedad $n-1$ dimensional).

El clasificador de margen máximo *(hard* o *soft)* en el que el producto escalar estándar ha sido sustituido por una función *kernel* genérica es una *support vector machine*.

Ahora, una vez presentada la teoría sobre las *support vector machines*, dedicaremos el siguiente apartado a la resolución de un ejemplo de ilustración de pequeño tamaño, que servirá para aclarar los conceptos que se han venido empleando.

Dado que, como se ha visto en la exposición teórica, en el procedimiento de entrenamiento de una *support vector machine* es necesario resolver un problema de programación matemática, recurriremos al entorno de programación R y, en concreto, a la función `ipop` del *package* `kernlab`.

Así, antes de resolver el ejemplo de ilustración, presentaremos brevemente la función `ipop` del *package* `kernlab`.

El paquete kernlab de R

La función `ipop` del paquete `kernlab` de R resuelve el siguiente problema de programación cuadrática:

Minimizar :

$$c^T x + \frac{1}{2} x^T H x$$

sujeto a:

$$b \leq A x \leq b + r$$

$$lower \leq x \leq upper$$

Para ello, la función a emplear es:

```
ipop(c,H,A,b,lower,upper,r)
```

La función `ipop` emplea el *software* LOQO, que es un paquete que emplea un método de punto interior para resolver problemas de optimización cuadrática y que fue creado por Vanderbei (1988).

Para adaptar la función `ipop` al problema dual asociado a un clasificador de margen máximo *soft* debemos ajustar los argumentos de dicha función:

- c es un vector columna de dimensión $l \times 1$ cuyos elementos son todos -1.
- $b = r = 0$
- *lower* es un vector columna de dimensión $l \times 1$ cuyos elementos son todos 0
- *upper* es un vector columna de dimensión $l \times 1$ cuyos elementos son todos igual al parámetro de coste C
- Si llamamos y al vector columna de dimensión $l \times 1$ que contiene las clases a las que

pertenecen los elementos de la muestra de aprendizaje, entonces la matriz $A = y^T$

- Si llamamos K a la matriz *kernel* que evalúa el *kernel* elegido en el conjunto de elementos de la muestra de aprendizaje, entonces la matriz H se obtiene modificando la matriz K del siguiente modo:

  - si el elemento *i*-ésimo y *j*-ésimo de la muestra de aprendizaje pertenecen a la misma clase entonces $H_{ij} = K_{ij}$

  - si, por el contrario, el elemento *i*-ésimo y el elemento *j*-ésimo de la muestra de aprendizaje pertenecen a clases diferentes entonces $H_{ij} = (-1) \times K_{ij}$

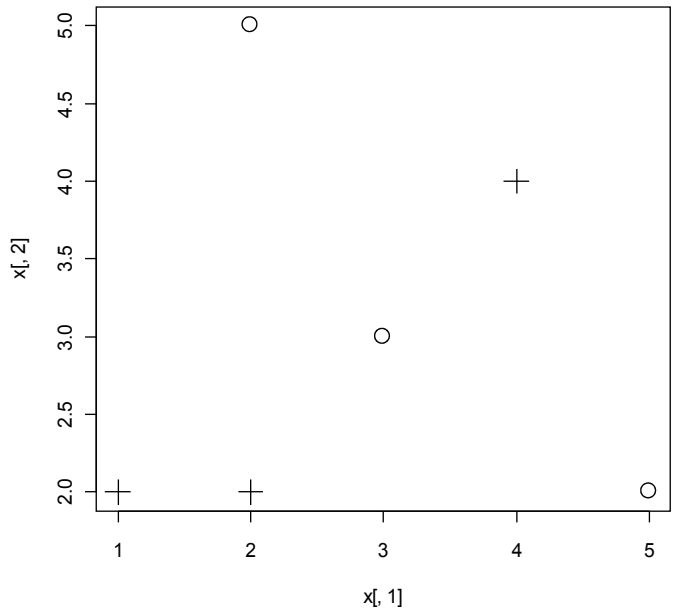
  De forma más compacta, $H = K \circ \left(y\, y^T \right)$, donde $\circ$ representa el *producto de Hadamard*[14] (o *producto de Schur*).

Ejemplo de ilustración de una support vector machine

Resolvamos con `ipop` el siguiente ejemplo de ilustración en el que se presenta una muestra de aprendizaje con dos clases no separables:

$$D = \{[(1,2)^T, +1], [(2,2)^T, +1], [(4,4)^T, +1], [(2,5)^T, -1], [(5,2)^T, -1], [(3,3)^T, -1]\}$$

La representación de estos seis elementos en el plano $\mathbb{R}^2$ es la siguiente:



14 El producto de Hadamard de dos matrices *A* y *B* de dimensión *m* x *n* es una matriz de dimensión *m* x *n* cuyos elementos son los productos de los correspondientes elementos de *A* y *B*.

donde las cruces representan los elementos de la clase +1 y los círculos los elementos de la clase -1. La representación gráfica muestra, claramente, que la intersección de las envolturas convexas de las clases no es vacía, por lo que el problema de determinar el clasificador de margen máximo *hard* carece de solución.

Consideremos, pues, la determinación del clasificador de margen máximo *soft* con parámetro de coste $C=5$.

Formulación dual del problema de optimización

Si llamamos X a la matriz que recoge las coordenadas de los elementos de la muestra de aprendizaje e y al vector columna que recoge las clases de dichos elementos, entonces el problema de optimización dual es el siguiente:

Minimizar:

$$-\sum_{i=1}^{6}\alpha_i+\frac{1}{2}\alpha^T\left[\left(X\,X^T\right)\circ\left(y\,y^T\right)\right]\alpha$$

sujeto a:

$$y^T\alpha=0$$

$$0\le\alpha_i\le C \qquad \forall i\in\{1,\ldots,6\}$$

Si empleamos la función `ipop` del *package* `kernlab` de R podemos proceder del modo recogido en el anexo R01.

El resultado es el siguiente:

```
> sv2

An object of class "ipop"
Slot "primal":
[1] 1.868238e-07 3.777777e+00 5.000000e+00 1.888889e+00 1.888889e+00 5.000000e+00

Slot "dual":
[1] -3.666666

Slot "how":
[1] "converged"
```

El resultado que proporciona la función `ipop` puede ser confuso ya que habla de Slot “dual” y de Slot “primal” y corremos el riesgo de confundir esta terminología con la que hemos venido empleando para referirnos a las diferentes formulaciones del problema de determinación del clasificador de margen máximo. Para evitar esta confusión hay que tener en cuenta que el Slot “primal” de la solución de `ipop` se refiere siempre al resultado del problema de programación que se haya planteado. Como en este caso se ha empleado la función `ipop` para resolver la formulación dual del problema, el Slot “primal” proporciona la solución de dicha formulación dual. Por su parte, el Slot “dual” de `ipop` proporciona la solución del problema dual del que se ha planteado a la

función. Así, en este caso, el Slot "dual" proporciona la solución (al menos parcial) a la formulación primal del problema de determinación del clasificador de margen máximo.

En definitiva, el resultado del problema de optimización dual es el siguiente:

- $\alpha_1^* = 0$
- $\alpha_2^* = 3{,}77$
- $\alpha_3^* = 5$
- $\alpha_4^* = 1{,}88$
- $\alpha_5^* = 1{,}88$
- $\alpha_6^* = 5$

En consecuencia, de acuerdo con lo que hemos visto en la exposición teórica, los elementos tercero (x=4, y=4) y sexto (x=3, y=3) están en el lado incorrecto de su correspondiente *supporting hyperplane*, el primer elemento (x=1, y=2) no juega ningún papel en la determinación de la frontera óptima de decisión y los *support vectors* de este ejemplo de ilustración son los elementos segundo, cuarto y quinto.

Para calcular las soluciones de la formulación primal del problema (es decir, la ecuación de la frontera óptima de decisión) basta con aplicar las fórmulas que hemos desarrollado con anterioridad:

$$w^* = \sum_{i=1}^{l} \alpha_i^* y_i x_i$$

y

$$b^* = w^{*T} x_p - y_p$$

siendo x_p las coordenadas de un *support vector* e y_p su clase

En R podemos resolverlo con los comandos recogidos en el anexo R02.

El resultado es:

```
> w
             x1         x2
[1,] -0.6666666 -0.6666666

> be
          [,1]
[1,] -3.666667
```

Es decir, la frontera de decisión óptima tiene como ecuación:

- $-\frac{2}{3}x_1-\frac{2}{3}x_2=-\frac{11}{3}$ es decir, $x_1+x_2=\frac{11}{2}$

Por su parte, los hiperplanos soporte tienen como ecuaciones respectivas:

- $-\frac{2}{3}x_1-\frac{2}{3}x_2=-\frac{11}{3}-1$, es decir, $x_1+x_2=7$ (para la clase -1)

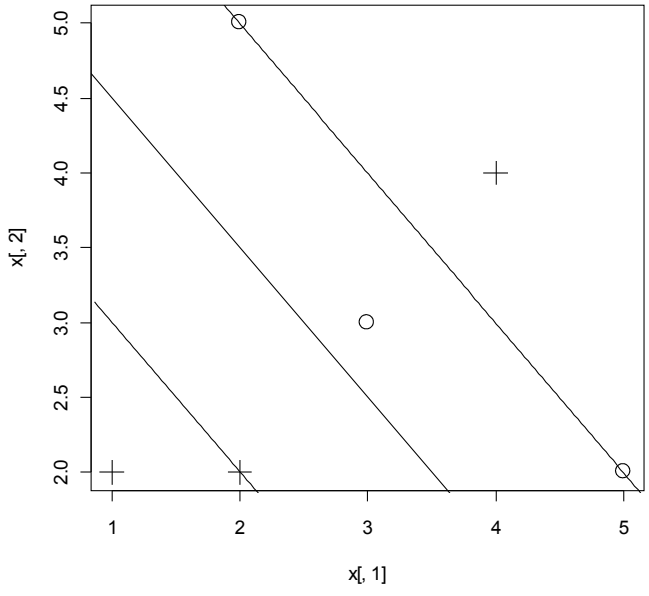
- $-\frac{2}{3}x_1-\frac{2}{3}x_2=-\frac{11}{3}+1$ es decir, $x_1+x_2=4$ (para la clase +1)

En el siguiente gráfico se presentan los elementos del ejemplo de ilustración con los hiperplanos soporte y la frontera óptima de decisión:

- El hiperplano soporte correspondiente a la clase +1 es la recta que pasa por el punto de coordenadas (2,2).

- El hiperplano soporte correspondiente a la clase -1 es la recta que pasa por los puntos de coordenadas respectivas (2,5) y (5,2).

- La frontera óptima de decisión es la recta paralela y equidistante de los dos hiperplanos soporte



Puede observarse que el elemento de la clase +1 situado en las coordenadas (4,4) se encuentra mal ubicado respecto a su hiperplano soporte y también respecto a la frontera de decisión óptima. No ocurre lo mismo con el elemento de la clase -1 situado en las coordenadas (3,3) que, a pesar de encontrarse mal ubicado en relación con su hiperplano soporte, resulta bien clasificado por la superficie óptima de decisión.

Formulación primal del problema de optimización

A continuación utilizaremos la función `ipop` para resolver el problema de determinación del clasificador de margen máximo *soft* en su formulación primal. Para ello, debemos recordar que dicha formulación primal del problema es:

$$min_{w,\xi,b}\left(\frac{1}{2}w^T w + C\sum_{i=1}^{l}\xi_i\right) \quad \text{con} \quad C>0$$

sujeto a las restricciones:

$$y_i(w^T x_i - b) + \xi_i - 1 \geq 0 \qquad \forall i \in \{1,\ldots,l\}$$

$$\xi_i \geq 0 \qquad \forall i \in \{1,\ldots,l\}$$

Para adaptar este problema al tipo de problemas que puede resolver la función `ipop` debemos hacer que:

- H sea una matriz diagonal de dimensión $(n+l+1)\times(n+l+1)$. Los n primeros elementos de la diagonal de la matriz serán 1 y los restantes serán 0.

- A sea una matriz de dimensión $l\times(n+1+l)$ cuyas n primeras columnas recogen las coordenadas de los individuos multiplicadas por la etiqueta de su clase, la siguiente columna recoge las etiquetas de los individuos multiplicadas por -1 y las l últimas columnas están formadas por una matriz identidad.

- c es un vector columna de dimensión $(n+1+l)\times 1$ cuyos $n+1$ primeros elementos son 0 y los restantes son C .

- b es un vector columna de dimensión $l\times 1$ cuyos elementos son todos iguales a 1

- r es un vector columna de dimensión $l\times 1$ cuyos elementos toman todos un valor muy elevado.

- *lower* es un vector columna de dimensión $(n+1+l)\times 1$ cuyos primeros $n+1$ elementos toman un valor muy reducido. Los restantes son 0.

- *upper* es un vector columna de dimensión $(n+1+l)\times 1$ cuyos elementos toman todos un valor muy elevado.

En definitiva, el problema de optimización primal asociado al ejemplo de ilustración puede resolverse mediante el código de R recogido en el anexo R03.

El resultado es:

```
An object of class "ipop"
Slot "primal":
[1] -6.666667e-01 -6.666667e-01 -3.666667e+00  1.212907e-09  1.852360e-08
[6]  2.666667e+00  4.334622e-09  4.137221e-09  6.666667e-01
```

```
Slot "dual":
[1] 4.675813e-08 3.777778e+00 5.000000e+00 1.888889e+00 1.888889e+00
[6] 5.000000e+00

Slot "how":
[1] "converged"
```

que confirma que:

- $w_1^* = \frac{-2}{3}$
- $w_2^* = \frac{-2}{3}$
- $b^* = \frac{-11}{3}$
- $\xi_1^* = \xi_2^* = \xi_4^* = \xi_5^* = 0$, por lo que los elementos 1,2,4 y 5 se encuentran adecuadamente situados en relación con su hiperplano soporte.
- $\xi_3^* = \frac{8}{3}$, por lo que el elemento tercero se encuentra en el lado inadecuado de la frontera de decisión (y, lógicamente, también en el lado inadecuado de su hiperplano soporte).
- $\xi_6^* = \frac{2}{3}$, por lo que el sexto elemento se encuentra en el lado incorrecto de su hiperplano soporte pero es clasificado correctamente por la frontera óptima de decisión.

Por último, es interesante destacar cómo la función `ipop` del package `kernlab` proporciona el mismo resultado para el `slot dual` del problema primal -`dual(sv1)`- que para el `slot primal` correspondiente al problema dual -`primal(sv2)`-. En el caso del `slot primal` del problema primal -`primal(sv1)`- y el `slot dual` del problema dual -`dual(sv2)`-, la coincidencia es sólo parcial, ya que este último tan sólo proporciona el valor de b^* . Estas coincidencias ponen de manifiesto, una vez más, la equivalencia entre el problema primal y el dual asociados al clasificador de margen máximo *soft*.

En el ejemplo de ilustración:

```
> dual(sv1)
[1] 4.675813e-08 3.777778e+00 5.000000e+00 1.888889e+00 1.888889e+00
[6] 5.000000e+00
> primal(sv2)
[1] 1.868238e-07 3.777777e+00 5.000000e+00 1.888889e+00 1.888889e+00
[6] 5.000000e+00

> primal(sv1)
[1] -6.666667e-01 -6.666667e-01 -3.666667e+00  1.212907e-09  1.852360e-08
[6]  2.666667e+00  4.334622e-09  4.137221e-09  6.666667e-01
> dual(sv2)
[1] -3.666666
```

Resolución del ejemplo de ilustración con una función *kernel*: *support vector machine*

Finalmente, para terminar con este pequeño ejemplo de ilustración, vamos a aprovechar la oportunidad que nos brinda la formulación dual del problema de clasificación de margen máximo *soft* de utilizar funciones *kernel.* En concreto, emplearemos el *kernel polinómico homogéneo de grado dos*, es decir:

$$k(x,z)=\langle x,z\rangle^2$$

El código de R[15] recogido en el anexo R04 sirve para resolver el problema de optimización dual asociado al clasificador de margen máximo *soft* con parámetro de coste $C=5$ y con *kernel* polinómico homogéneo de grado 2.

El resultado es el siguiente:

```
An object of class "ipop"
Slot "primal":
[1] 2.653958e+00 1.267490e-08 2.742778e+00 3.879367e-01 8.799781e-03 5.000000e+00

Slot "dual":
[1] -1.266666

Slot "how":
[1] "converged"
```

El resultado de este problema de optimización es:

- $\alpha_1^*=2{,}654$
- $\alpha_2^*=0$
- $\alpha_3^*=2{,}743$
- $\alpha_4^*=0{,}388$
- $\alpha_5^*=0{,}008$
- $\alpha_6^*=5$

En resumen, los elementos 1, 3, 4 y 5 son *support vectors* y junto con el 2 (que no lo es) se encuentran correctamente situados respecto a su hiperplano soporte (aunque ahora esté hiperplano ya no es lineal en el espacio inicial aunque sí en el *feature space*). El elemento 6 se encuentra incorrectamente situado en relación con su hiperplano soporte.

En cuanto a la ecuación del hiperplano soporte, ésta es:

15 Si comparamos este código con el que se empleó para la formulación dual del problema en el caso lineal, veremos que la única diferencia estriba en la definición de la matriz *H* que, en esta ocasión, se define a partir de la matriz *kernel.*

$$\sum_{i=1}^{l} \alpha_i^* y_i k(x_i, x) = \sum_{i=1}^{l} \alpha_i^* y_i k(x_i, x_p) - y_p$$

Para obtenerla[16], debemos evaluar la función *kernel* en distintas parejas[17]. Tenemos que:

- $k(\boldsymbol{x_1}, x) = x_1^2 + 4x_2^2 + 4x_1x_2$
- $k(\boldsymbol{x_3}, x) = 16x_1^2 + 16x_2^2 + 32x_1x_2$
- $k(\boldsymbol{x_4}, x) = 4x_1^2 + 25x_2^2 + 20x_1x_2$
- $k(\boldsymbol{x_5}, x) = 25x_1^2 + 4x_2^2 + 20x_1x_2$
- $k(\boldsymbol{x_6}, x) = 9x_1^2 + 9x_2^2 + 18x_1x_2$

Podemos elegir como *support vector* el primer elemento, cuya clase es +1. En consecuencia:

- $k(x_1, x_1) = 25$
- $k(x_3, x_1) = 144$
- $k(x_4, x_1) = 144$
- $k(x_5, x_1) = 81$
- $k(x_6, x_1) = 81$

Operando con todos estos resultados, la ecuación de la superficie de decisión resulta ser:

$$-0{,}23333532x_1^2 - 0{,}23333662x_2^2 + 0{,}44999838x_1x_2 = -1{,}26668506$$

y las de los hiperplanos soporte:

$$-0{,}23333532x_1^2 - 0{,}23333662x_2^2 + 0{,}44999838x_1x_2 = -0{,}26668506$$

$$-0{,}23333532x_1^2 - 0{,}23333662x_2^2 + 0{,}44999838x_1x_2 = -2{,}26668506$$

que corresponden a tres elipses centradas en el origen, rotadas un ángulo de $\pi/4$ en sentido antihorario y cuyos semiejes respectivos son:

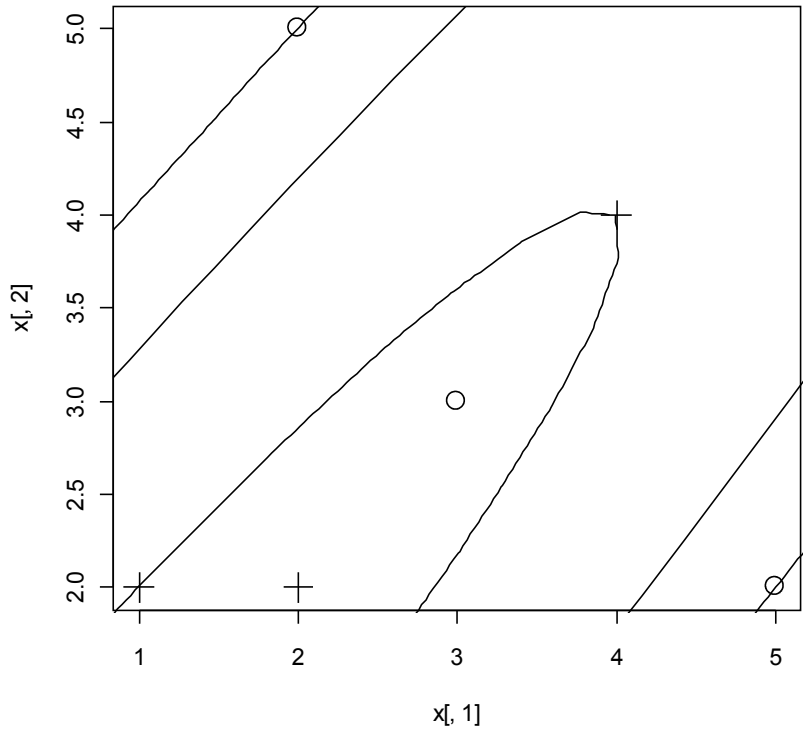
- 12,3263 y 1,6624

16 Conviene no confundir los elementos $\boldsymbol{x_1}$ y $\boldsymbol{x_2}$ con las coordenadas de x, que son x_1 y x_2. Para tratar de evitar esa confusión se ha adoptado una tipografía diferente en este segmento del texto. En lo sucesivo, se retomará la tipografía habitual.

17 En realidad estamos determinando la imagen por el *embedding* $\phi : x \to k(x, \cdot)$ para los elementos de la muestra a los que se les ha asignado en la solución un valor de α distinto de 0, es decir, todos excepto el segundo.

- 5,6559 y 0,7628
- 16,4891 y 2,2238

Podemos dibujar estas elipses sobre el gráfico original con el código del anexo R05:



- El hiperplano soporte de la clase +1 es la elipse interior, que pasa por los puntos de coordenadas (1,2) y (4,4).
- El hiperplano soporte de la clase -1 es la elipse exterior, que pasa por los puntos de coordenadas (2,5) y (5,2)
- La superficie óptima de decisión es la elipse intermedia

Con este resultado queda patente que la representación de los *hiperplanos soporte* en el *input space* no es lineal. Se trata, en este caso de sendas elipses

Se aprecia claramente cómo con este *kernel* polinómico homogéneo de grado 2, los elementos de coordenadas (1,2), (4,4), (2,5) y (5,2) son *support vectors* mientras que el elemento de coordenadas (2,2) no lo es y el de coordenadas (3,3) tampoco, encontrándose este último no sólo en el lado incorrecto de su hiperplano soporte (en el interior de la elipse exterior) sino también mal clasificado por la frontera de decisión (en el interior de la elipse intermedia).

Con este ejemplo de ilustración se ha perseguido fijar los conceptos que sobre los clasificadores de

margen máximo, en general, y sobre las máquinas de vector soporte, en particular, se han expuesto en la parte teórica. En la parte final del documento se presentará una nueva aplicación de las *support vector machines*, en ese caso para analizar una base de datos real. No obstante, antes de ver esa aplicación real, dedicaremos la siguiente parte del trabajo a la exposición teórica de los fundamentos de la integración de las funciones *kernel* en el ámbito del aprendizaje no supervisado y, más en concreto, presentaremos la técnica del *kernel PCA* o análisis de componentes principales basado en funciones *kernel*.

5. Aplicación de las funciones *kernel* al aprendizaje no supervisado: *kernel PCA*

La metodología basada en funciones *kernel* que se viene describiendo para el caso del aprendizaje supervisado (y en concreto, en el ámbito de los clasificadores de margen máximo) es también de aplicación en las tareas de aprendizaje estadístico de tipo no supervisado, que se caracterizan por el hecho de que los elementos de la muestra de aprendizaje carecen de una etiqueta que permita clasificarlos.

Entre estas tareas de aprendizaje no supervisado cabe citar la *clasificación automática* (o análisis clúster), cuyo objetivo es establecer una tipología en el conjunto de elementos de la muestra, de forma que los grupos resultantes sean homogéneos internamente y muy diferentes entre ellos, y también el *análisis de componentes principales*, mediante el cual se pretende encontrar un subespacio del espacio original en el que se encuentran representados los elementos, de dimensión reducida pero que, simultáneamente, permita una adecuada representación de los datos, es decir, suponga la mínima pérdida de información.

En este apartado del trabajo expondremos los fundamentos teóricos correspondientes a la aplicación de las funciones *kernel* al análisis de componentes principales, metodología que recibe el nombre de *kernel PCA*.

En cualquier caso, como se expondrá de modo somero durante la resolución de algunos de los ejemplos de ilustración, cualquier procedimiento de aprendizaje no supervisado que tome como punto de partida una matriz de distancias (y este es el caso, por ejemplo, del análisis clúster) puede ser fácilmente *kernelizado*, esto es, reprogramado para poder adaptarlo al uso de las funciones *kernel*.

Comenzamos, al igual que en el caso del aprendizaje supervisado, por presentar los fundamentos del análisis de componentes principales tradicional. Veremos dos formulaciones:

- la primera, basada en la matriz de covarianzas entre las variables, y que podríamos denominar, siguiendo la terminología empleada en la sección anterior, formulación primal

- la segunda, basada en la matriz de los productos escalares entre las coordenadas de los individuos, y que podríamos denominar formulación dual. Será esta segunda formulación la que facilite la incorporación de las funciones *kernel* al ámbito del análisis de componentes principales, dando lugar a la técnica conocida como *kernel PCA*.

Análisis de componentes principales y descomposición espectral de una matriz simétrica

Como se ha comentado, el análisis de componentes principales es una técnica de aprendizaje no supervisado cuyo objetivo es encontrar un subespacio del espacio original en el que se encuentran representados los datos, que debe cumplir dos condiciones:

- Ser de dimensión reducida

- Recoger una gran parte de la variabilidad original de los datos

En este apartado presentaremos los fundamentos del análisis de componentes principales tradicional. En primer lugar, lo resolveremos como un problema de descomposición espectral de la matriz de covarianzas entre las variables medidas (lo que llamaremos formulación primal) para, a continuación, tratarlo como un problema de descomposición espectral de la matriz de los productos escalares entre las coordenadas de los individuos (formulación dual).

El análisis de componentes principales a partir de la matriz de covarianzas

Supongamos que disponemos de una muestra de aprendizaje no etiquetada:

$$D=\{x_1,\ldots,x_l\} \quad \text{con} \quad x_i\in\mathbb{R}^k \qquad \forall i\in\{1,\ldots,l\}$$

Supongamos, asimismo, que los datos han sido centrados, es decir, que el centro de masas de D coincide con el origen de $\mathbb{R}^k$.

Podemos recoger todos estos datos en una matriz X de dimensión $l\times k$ cuyas filas son los $x_i^T \quad \forall i\in\{1,\ldots,l\}$. Diremos que en la muestra de aprendizaje hay l *individuos* y k *variables*. El elemento genérico de la matriz X será x_{ij}, es decir, el valor que el *i*-ésimo individuo toma en la *j*-ésima variable.

Consideremos ahora la matriz X^TX. Su elemento genérico es[18]:

$$(X^TX)_{jj'}=\sum_{i=1}^{l} x_{ij}x_{ij'}=l\,cov(x_{\cdot j},x_{\cdot j'})$$

por lo que podemos decir que:

$$X^TX=lC$$

siendo C la *matriz de covarianzas*, de dimensión $k\times k$, entre las variables de la muestra de aprendizaje[19].

La matriz X^TX es de utilidad para calcular la dispersión de las observaciones en una dirección determinada del espacio vectorial $\mathbb{R}^k$. En efecto, si $u\in\mathbb{R}^k$ es un vector unitario, resulta que el módulo de la proyección del individuo *i*-ésimo sobre el vector unitario $u\in\mathbb{R}^k$ viene dado por

$$x_i^Tu$$

por lo que si deseamos obtener los módulos de las proyecciones de todos los individuos en la

18 Debe tenerse en cuenta que las variables están centradas y, por ese motivo, el promedio de los productos entre las puntuaciones de los individuos en una pareja de variables es igual a la covarianza entre dichas variables.

19 Es habitual referirse a la matriz X^TX como *matriz de covarianzas* aunque, en puridad, se diferencian en una constante. Este abuso del lenguaje no es grave ya que ambas matrices comparten, como se verá más adelante, los vectores propios y los valores propios de una y otra están relacionados por una constante de proporcionalidad igual al número de individuos.

dirección del vector $u \in \mathbb{R}^k$, podemos hacerlo calculando el vector l dimensional:

$$X u$$

Como los datos están centrados resulta que:

$$\frac{1}{l}\sum_{i=1}^{l}(X u)_i = \frac{1}{l}\sum_{i=1}^{l} x_i^T u = 0$$

y, en consecuencia:

$$\sum_{i=1}^{l}[(X u)_i]^2 = (X u)^T (X u) = u^T (X^T X) u$$

es decir, la variabilidad de las proyecciones de los individuos de la muestra de aprendizaje en la dirección del vector u viene dada por:

$$u^T (X^T X) u$$

Si deseamos encontrar la dirección $u^* \in \mathbb{R}^k$ que hace máxima la expresión anterior, podemos apoyarnos en el hecho de que la matriz $X^T X$ es simétrica y semidefinida positiva. Por ser simétrica, $X^T X$ es ortogonalmente diagonalizable; por ser semidefinida positiva, todos sus valores propios son no negativos. En consecuencia, existe una matriz ortogonal U, de dimensión $k \times k$, y una matriz diagonal Λ de dimensión $k \times k$ y cuyos elementos son todos no negativos, que cumplen:

$$X^T X = U \Lambda U^T$$

o, de modo equivalente:

$$X^T X = \lambda_1 u_1 u_1^T + \ldots + \lambda_k u_k u_k^T$$

donde los $\lambda_1, \ldots, \lambda_k$ son los elementos de la diagonal de Λ (es decir, los valores propios, que son todos no negativos, de $X^T X$) y los $u_1, \ldots, u_k$ son los vectores propios y unitarios asociados a los valores propios anteriores, es decir, las columnas de la matriz U.

Así, podemos expresar la variabilidad (o inercia) a lo largo del vector unitario $u \in \mathbb{R}^k$ de este modo:

$$u^T (X^T X) u = u^T (\lambda_1 u_1 u_1^T + \ldots + \lambda_k u_k u_k^T) u$$

y, desarrollando:

$$u^T (X^T X) u = [u^T (u_1 u_1^T) u]\lambda_1 + \ldots + [u^T (u_k u_k^T) u]\lambda_k = \sum_{j=1}^{k} [u^T (u_j u_j^T) u]\lambda_j$$

pero

$$u^T(u_j u_j^T)u = (u^T u_j)(u_j^T u) = \|u_j^T u\|^2 = \|p_{u_j}(u)\|^2$$

luego

$$u^T(X^T X)u = \sum_{j=1}^{k} \|p_{u_j}(u)\|^2 \lambda_j$$

Por otra parte:

$$\sum_{j=1}^{k} \|p_{u_j}(u)\|^2 = \begin{pmatrix} \|p_{u_1}(u)\| & \dots & \|p_{u_k}(u)\| \end{pmatrix} \begin{pmatrix} \|p_{u_1}(u)\| \\ \dots \\ \|p_{u_k}(u)\| \end{pmatrix} = \begin{pmatrix} u^T u_1 & \dots & u^T u_k \end{pmatrix} \begin{pmatrix} u_1^T u \\ \dots \\ u_k^T u \end{pmatrix}$$

Por lo tanto,

$$\sum_{j=1}^{k} \|p_{u_j}(u)\|^2 = u^T U\, U^T u = u^T u = 1$$

donde la segunda igualdad se cumple por ser U una matriz ortogonal y la tercera por ser u un vector unitario.

Como, obviamente,

$$\|p_{u_j}(u)\|^2 \geq 0 \qquad \forall\, j \in \{1, \dots, k\}$$

y, según acabamos de demostrar,

$$\sum_{j=1}^{k} \|p_{u_j}(u)\|^2 = 1$$

concluimos que la variabilidad de las proyecciones de los individuos sobre el vector $u \in \mathbb{R}^k$ es una *combinación lineal convexa* de los valores propios de la matriz $X^T X$

$$u^T(X^T X)u = \sum_{j=1}^{k} \|p_{u_j}(u)\|^2 \lambda_j$$

Estamos ya en condiciones de determinar cuál es la dirección del espacio $\mathbb{R}^k$ definida por un vector $u^* \in \mathbb{R}^k$ a lo largo del cual la variabilidad de las proyecciones de los individuos es máxima. Si deseamos maximizar

$$u^T(X^T X)u = \sum_{j=1}^{k} \|p_{u_j}(u)\|^2 \lambda_j$$

y suponiendo que los $\lambda_1, \dots, \lambda_k$ están ordenados en orden decreciente, nos interesará que el coeficiente de λ_1 en la combinación lineal convexa anterior sea máximo, es decir, que:

$$\|p_{u_1}(u)\|^2 = u^T(u_1 u_1^T)u = 1$$

Naturalmente esto se consigue haciendo que:

$$u^* = u_1$$

siendo u_1 el vector propio unitario de $X^T X$ asociado al máximo valor propio, λ_1 .

Nótese que al hacer $u = u_1$ se consigue que:

$$\|p_{u_j}(u)\|^2 = u^T(u_j u_j^T)u = 0 \quad \forall\, j \neq 1$$

Si ahora quisiéramos encontrar una dirección $u^{**} \in \mathbb{R}^k$, ortogonal a $u^* = u_1$ y a lo largo de la cual la variabilidad de las proyecciones sea máxima, debemos tener en cuenta que el coeficiente de λ_1 en

$$u^T(X^T X)u = [u^T(u_1 u_1^T)u]\lambda_1 + \ldots + [u^T(u_k u_k^T)u]\lambda_k = \sum_{j=1}^{k} [u^T(u_j u_j^T)u]\lambda_j$$

se anulará, dada la ortogonalidad entre $u^* = u_1$ y u^{**} que hemos exigido. Por tanto, la elección óptima para maximizar la variabilidad sujeta a la restricción de que esta dirección debe ser ortogonal a $u^* = u_1$ se consigue al hacer máximo el coeficiente del segundo mayor valor propio.

Naturalmente, esto se consigue haciendo que:

$$u^{**} = u_2$$

siendo u_2 el vector propio unitario de $X^T X$ asociado al segundo mayor valor propio.

Nuevamente, al hacer $u = u_2$ se consigue que:

$$\|p_{u_j}(u)\|^2 = u^T(u_j u_j^T)u = 0 \quad \forall\, j \neq 2$$

En general, si deseamos encontrar un vector unitario $u^{j(*)} \in \mathbb{R}^k$ a lo largo del cual se maximice la variabilidad de las proyecciones de los individuos y que sea ortogonal a $u_1, \ldots, u^{j-1(*)}$ debemos considerar el vector propio u_j asociado al *j*-ésimo mayor valor propio de $X^T X$.

Es decir,

$$u^{j(*)} = u_j$$

Además, la variabilidad de las proyecciones de los individuos en cada una de estas direcciones viene dada por:

$$u_j{}^T(X^T X)u_j = [u_j{}^T(u_1 u_1^T)u_j]\lambda_1 + \ldots + [u_j{}^T(u_k u_k^T)u_j]\lambda_k = \lambda_j$$

ya que

$$u_j^T (u_m u_m^T) u_j = 0 \quad \text{si} \quad m \neq j \quad \text{y} \quad u_j^T (u_m u_m^T) u_j = 1 \quad \text{si} \quad m = j$$

Se demuestra fácilmente (y, además, es intuitivamente aparente) que el subespacio vectorial de $\mathbb{R}^k$ de dimensión *m* que recoge una mayor proporción de la variabilidad de los individuos es el generado por los *m* vectores propios unitarios de $X^T X$ asociados a los *m* mayores valores propios de dicha matriz. Además, la variabilidad recogida vendrá dada por

$$\sum_{j=1}^{m} \lambda_j$$

Sirva esta rápida introducción para ilustrar el hecho de que el análisis de componentes principales (es decir, la búsqueda de un subespacio vectorial de dimensión reducida que recoja la mayor parte de la variabilidad del conjunto de individuos) es, en esencia, un problema de descomposición espectral de la matriz $X^T X$ o, lo que es equivalente, de la matriz de covarianzas C, ya que:

- si $u \in \mathbb{R}^k$ es un vector propio de $X^T X$ asociado a λ entonces $X^T X u = \lambda u$ y, en consecuencia:

$$C u = \frac{1}{l} X^T X u = \frac{1}{l} \lambda u$$

  es decir, u también es vector propio de C asociado al valor propio λ / l.

- si $u \in \mathbb{R}^k$ es un vector propio de C asociado a λ entonces $C u = \lambda u$ y, en consecuencia:

$$X^T X u = l C u = l \lambda u$$

  es decir, u también es vector propio de $X^T X$ asociado al valor propio $l \lambda$.

En otras palabras, las matrices $X^T X$ y C comparten vectores propios y los valores propios de una se pueden obtener muy fácilmente a partir de los valores propios de la otra.

De esta forma, concluimos la exposición de la formulación primal del análisis de componentes principales tradicional. La siguiente sección se dedica a exponer una formulación equivalente, basada en la matriz de los productos escalares entre las coordenadas de los individuos. Será esta formulación la que servirá como puerta de entrada a las funciones *kernel*, es decir, a la *kernelización* del análisis de componentes principales.

El análisis de componentes principales a partir de la matriz de los productos escalares. *Kernelización* del análisis de componentes principales

Tal como se ha expuesto, basta con la matriz de covarianzas entre las variables medidas en el colectivo de individuos para obtener un subespacio de máxima variabilidad y mínima dimensión. En efecto, la matriz C o, en su defecto, la matriz $X^T X$ son suficientes para ese fin.

Sin embargo, existen otras matrices que también permiten el cálculo de dicho subespacio de variabilidad máxima y dimensión mínima.

En efecto, consideremos la matriz $X X^T$ (de dimensión $l \times l$). Su elemento genérico será:

$$(X X^T)_{ii'} = \sum_{j=1}^{k} x_{ij} x_{i'j} = \langle x_i, x_{i'} \rangle$$

es decir, el producto escalar entre las coordenadas de los elementos (individuos) x_i y $x_{i'}$.

Veamos cómo esta matriz (que sólo contiene los productos escalares entre los individuos y que, por tanto, podremos utilizar para generalizar dicho producto escalar a una función *kernel* y aplicar esta metodología) permite también la determinación del subespacio vectorial de $\mathbb{R}^k$ de variabilidad máxima y dimensión mínima para el conjunto de datos analizado.

En primer lugar, por ser $X X^T$ una matriz simétrica y semidefinida positiva, es ortogonalmente diagonalizable y sus valores propios son todos no negativos.

Supongamos que $w \in \mathbb{R}^l$ es un vector propio y unitario de $X X^T$ asociado al valor propio λ . Entonces resulta que:

$$X X^T w = \lambda w$$

y si premultiplicamos ambos miembros de la ecuación anterior por X^T resulta:

$$(X^T X) X^T w = \lambda X^T w$$

es decir, si w es vector propio y unitario de $X X^T$, entonces $X^T w$ es vector propio (aunque no necesariamente unitario) de $X^T X$. En ambos casos, el valor propio asociado al vector propio es el mismo.

Si queremos obtener un vector propio de $X^T X$ que, además, sea unitario basta con encontrar un valor $k \in \mathbb{R}$ tal que

$$\| k X^T w \|^2 = (k X^T w)^T (k X^T w) = k^2 w^T X X^T w = k^2 \lambda w^T w = k^2 \lambda = 1$$

En resumen, si $w \in \mathbb{R}^l$ es vector propio y unitario de $X X^T$ asociado al valor propio λ , entonces:

$$\frac{1}{\sqrt{\lambda}} X^T w \in \mathbb{R}^k$$

es vector propio y unitario de $X^T X$ asociado al mismo valor propio λ .

Hemos visto ya que las proyecciones de todos los individuos en la dirección de un vector unitario $u \in \mathbb{R}^k$ se calculan haciendo:

$$X u$$

En consecuencia, las proyecciones de todos los individuos sobre el vector unitario

$$\frac{1}{\sqrt{\lambda}} X^T w \in \mathbb{R}^k$$

se calcularán del siguiente modo:

$$X\left(\frac{1}{\sqrt{\lambda}} X^T w\right) = \frac{1}{\sqrt{\lambda}} X X^T w \in \mathbb{R}^l$$

por lo que resulta patente que para calcular estas proyecciones es suficiente con conocer la matriz $X X^T$ y su descomposición espectral (que nos proporciona w y λ)

Lo anterior pone de manifiesto que:

- Es posible determinar los valores propios de $X^T X$ (o de C) y, en consecuencia, la dispersión de las proyecciones de los individuos sobre las direcciones de máxima variabilidad del espacio vectorial $\mathbb{R}^k$ a partir, exclusivamente, de la información recogida en la matriz $X X^T$ que, como ya se ha mencionado, se refiere exclusivamente a los productos escalares entre los individuos que conforman la muestra.

- Asimismo, es posible calcular las proyecciones de los individuos en dichas direcciones de máxima variabilidad a partir de la información recogida en la matriz $X X^T$.

De este modo, queda abierta la puerta a la sustitución del producto escalar estándar $\langle x_i, x_{i'} \rangle$, por una función *kernel* $k(x_i, x_{i'})$ que dote al análisis de componentes principales tradicional de todo un conjunto de nuevas herramientas que le permitan superar su inherente linealidad.

En efecto, si sustituimos la matriz $X X^T$ de las fórmulas anteriores por una matriz *kernel* K calculada evaluando cierta función *kernel* $k(x_i, x_{i'})$ en el conjunto de individuos que componen la muestra, estaremos, en realidad, realizando un análisis de componentes principales sobre el conjunto de datos original transformado por el *embedding* (implícito y que no necesitaremos conocer) asociado a la función *kernel*.

Es decir, el conjunto de datos original

$$D = \{x_1, \ldots, x_l\} \quad \text{con} \quad x_i \in \mathbb{R}^k \qquad \forall i \in \{1, \ldots, l\}$$

será sustituido por

$$\phi(D) = \{\phi(x_1), \ldots, \phi(x_l)\} \quad \text{con} \quad \phi(x_i) \in F \qquad \forall i \in \{1, \ldots, l\}$$

Así, la matriz que recogía los datos originales

$$X \quad \text{de dimensión} \quad l \times k$$

será sustituida por

$$\phi(X)$$

matriz de l filas (y número de columnas que dependerá del *kernel* elegido).

Las filas de $\phi(X)$ serán:

$$\phi(x_i)^T \qquad \forall i \in \{1,\ldots,l\}$$

Así, la matriz de covarianzas en este *feature space* tendrá dimensión dependiente del *kernel* elegido, pero, en cualquier caso, será de la forma:

$$C = \frac{1}{l}[\phi(X)]^T[\phi(X)]$$

Como se ha visto en anteriores apartados, es la descomposición espectral de esta matriz C la que proporciona las direcciones de máxima variabilidad de las proyecciones de los individuos en el *feature space*. Lamentablemente, en el caso de ciertas funciones *kernel* (por ejemplo, si empleáramos un *kernel gaussiano*) la dimensión de la matriz C sería infinita.

Afortunadamente, podemos acceder a los valores propios de C así como a las proyecciones de los individuos sobre las direcciones de máxima variabilidad mediante la descomposición espectral de la matriz

$$[\phi(X)][\phi(X)]^T$$

cuya dimensión es $l \times l$ y cuyo elemento genérico es:

$$[\phi(X)][\phi(X)]^T_{i,i'} = \langle \phi(x_i), \phi(x_{i'}) \rangle = k(x_i, x_{i'}) = K_{i,i'}$$

En definitiva, la matriz $[\phi(X)][\phi(X)]^T$ es la matriz *kernel* K.

En consecuencia, para obtener las direcciones de máxima inercia en el *feature space* así como el valor de la variabilidad de las proyecciones de las observaciones en dichas direcciones basta con descomponer espectralmente la matriz *kernel* K.

En cuanto a las proyecciones de los individuos en las direcciones de máxima variabilidad del *feature space*, supongamos que $w \in \mathbb{R}^l$ es un vector propio y unitario de $[\phi(X)][\phi(X)]^T = K$ asociado al valor propio λ y que, en consecuencia

$$\frac{1}{\sqrt{\lambda}}\phi(X)^T w$$

es un vector propio y unitario de $[\phi(X)]^T[\phi(X)]$ asociado al mismo valor propio.

Las proyecciones de los individuos $\phi(D)$ sobre esta dirección vendrán dadas por:

$$\frac{1}{\sqrt{\lambda}}\phi(X)\phi(X)^T w = \frac{1}{\sqrt{\lambda}} K w$$

por lo que, una vez más, se pone de manifiesto que es innecesario conocer cuál es el *embedding* concreto asociado a la función *kernel* elegida.

Finalmente hay que tener en cuenta que hemos supuesto que el conjunto de datos está centrado en el *input space*, pero no nos hemos asegurado de que ocurra lo mismo cuando los datos se incrustan en el *feature space* a través del *embedding.* Para asegurarnos de que los datos incrustados están centrados en el *feature space,* antes de proceder a la diagonalización de la matriz K, debemos aplicar el procedimiento de transformación que vimos en una sección anterior y que matricialmente se expresa del siguiente modo:

$$\hat{K} = K - \frac{1}{l} K j j^T - \frac{1}{l} j j^T K + \frac{1}{l^2} j^T K j j j^T$$

donde j es un vector de unos.

Una vez expuesto el fundamento teórico del *kernel PCA*, a continuación se ponen en práctica todos estos conceptos es un ejemplo de tamaño reducido, que permite efectuar los cálculos necesarios de manera sencilla.

Ejemplo de ilustración del análisis de componentes principales y del *kernel PCA*

Supongamos que disponemos de una tabla de datos con $l = 5$ individuos para los que se han medido $k = 3$ variables. Los datos se recogen en la siguiente tabla:

Individuo	x1	x2	x3
1	1	1	8
2	2	4	7
3	2	7	5
4	3	3	6
5	5	1	9

A partir de estos datos se desea realizar un análisis de componentes principales no normado (es decir, con carácter previo al análisis los datos se centrarán pero no se tipificarán).

Como se ha visto en apartados anteriores, este problema es, en esencia, reducible a la descomposición espectral de la matriz de covarianzas entre las variables (o de la matriz $X^T X$). Análogamente (y esta es la vía más interesante para introducir posteriormente las funciones *kernel* y, en consecuencia, el llamado *kernel PCA*), el problema puede resolverse también descomponiendo espectralmente la matriz de los productos escalares entre los vectores asociados a los individuos.

En la resolución del ejemplo de ilustración trabajaremos:

- En primer lugar, con la matriz $X^T X$, que tiene idénticos vectores propios que la matriz de covarianzas y valores propios iguales a los de la matriz de covarianzas multiplicados por el número de individuos, l.

- En segundo lugar, con la matriz de productos escalares $X\,X^T$.

Veremos cómo a partir de cualquiera de estas dos matrices es posible calcular tanto las coordenadas de los individuos en las direcciones de máxima variabilidad como la propia variabilidad existente en cada una de esas direcciones llegando, naturalmente, a idénticos resultados.

El código de R recogido en el anexo R06 realiza los cálculos necesarios tras leer los datos, que se encuentran en el archivo (de valores separados por comas) *DatosKernelPCA.csv*.

La matriz de los datos centrados es:

```
> X
       x1   x2 x3
[1,] -1.6 -2.2  1
[2,] -0.6  0.8  0
[3,] -0.6  3.8 -2
[4,]  0.4 -0.2 -1
[5,]  2.4 -2.2  2
```

La matriz $X^T X$ es:

```
> C
     x1    x2  x3
x1  9.2  -4.6   4
x2 -4.6  24.8 -14
x3  4.0 -14.0  10
```

La descomposición espectral de la matriz de $X^T X$ proporciona los siguientes vectores propios unitarios (por columnas)

```
> W_C
           [,1]       [,2]       [,3]
[1,]  0.2298742 0.96067318 -0.1557719
[2,] -0.8298023 0.27710340  0.4843984
[3,]  0.5085134 0.01790925  0.8608678
```

y los siguientes valores propios:

```
> LAMBDA_C
[1] 34.653683  7.947713  1.398604
```

Las proyecciones de los individuos sobre las direcciones expresadas por los vectores propios unitarios (obtenidas a partir de la descomposición espectral de la matriz $X^T X$) son :

```
> F_C
           [,1]       [,2]       [,3]
[1,]  1.9662799 -2.1287953  0.0444265
[2,] -0.8017664 -0.3547212  0.4809819
[3,] -4.3082003  0.4407705  0.2124414
[4,] -0.2506033  0.3109393 -1.0200562
[5,]  3.3942901  1.7318066  0.2822065
```

Análogamente, podemos proceder a través de la matriz de productos escalares, que es:

```
> K
     [,1] [,2]  [,3] [,4]  [,5]
```

```
[1,]  8.4 -0.8  -9.4 -1.2   3.0
[2,] -0.8  1.0   3.4 -0.4  -3.2
[3,] -9.4  3.4  18.8  1.0 -13.8
[4,] -1.2 -0.4   1.0  1.2  -0.6
[5,]  3.0 -3.2 -13.8 -0.6  14.6
```

La descomposición espectral de esta matriz de productos escalares proporciona los siguientes vectores propios y unitarios (por columnas)

```
W_K
            [,1]       [,2]        [,3]       [,4]       [,5]
[1,]  0.33401859  0.7551145  0.03756598  0.0000000 -0.5628698
[2,] -0.13619876  0.1258247  0.40670671  0.8871049  0.1151199
[3,] -0.73184850 -0.1563477  0.17963533 -0.0905209 -0.6320524
[4,] -0.04257083 -0.1102947 -0.86253506  0.4345004 -0.2307932
[5,]  0.57659949 -0.6142969  0.23862704  0.1267293 -0.4660141
```

y estos valores propios (que naturalmente coinciden con los de la matriz $X^T X$):

```
> LAMBDA_K
[1]  3.465368e+01  7.947713e+00  1.398604e+00  3.601205e-15 -8.563399e-16
```

A partir de la matriz de productos escalares y de sus valores y vectores propios pueden calcularse las proyecciones de los individuos sobre los componentes principales:

```
> F_K
           [,1]       [,2]       [,3]          [,4] [,5]
[1,]  1.9662799  2.1287953  0.0444265  6.963479e-09  NaN
[2,] -0.8017664  0.3547212  0.4809819  1.184790e-08  NaN
[3,] -4.3082003 -0.4407705  0.2124414 -1.139848e-08  NaN
[4,] -0.2506033 -0.3109393 -1.0200562 -4.148378e-08  NaN
[5,]  3.3942901 -1.7318066  0.2822065  4.361016e-08  NaN
```

que coinciden, salvo los signos, con las calculadas a partir de la matriz $X^T X$. Las diferencias en los signos se deben a la arbitrariedad en la elección del sentido de los vectores propios asociados a los diferentes valores propios y, por tanto, carecen de importancia. Las dos últimas columnas de la anterior matriz recogen valores que, o bien son nulos, o bien carecen de sentido. Esto se debe a que la dimensión original del problema es 3 (hay tres variables recogidas para el conjunto de individuos).

Resolución del ejercicio de ilustración con un *kernel* polinómico homogéneo de grado dos

Con este ejemplo de ilustración se ha puesto de manifiesto que el resultado obtenido a través de la descomposición espectral de la matriz de covarianzas puede también alcanzarse mediante la descomposición espectral de la matriz de productos escalares. Dicho de otra forma, el procedimiento basado en la diagonalización de la matriz de productos escalares constituye la *kernelización* del algoritmo del análisis de componentes principales basado en la descomposición espectral de la matriz de covarianzas ya que proporciona un método tal que la única información que necesita el algoritmo para alcanzar su objetivo es la relativa a los productos escalares entre los individuos de la muestra.

Así, si sustituimos los productos escalares entre los individuos por una matriz *kernel* resultado de evaluar sobre todas las parejas de individuos una función *kernel* válida, estaremos efectuando un *kernel PCA*, es decir, un análisis de componentes principales tradicional sobre el conjunto de las imágenes (mediante cierto *embedding)* de los datos originales.

Naturalmente, conviene efectuar este *kernel PCA* mediante la descomposición espectral de la matriz de productos escalares en el *feature space,* ya que esta es la matriz *kernel* ya calculada. Sin embargo, dado que el reducido tamaño del ejemplo de ilustración lo permite, y a efectos de mostrar con mayor detalle el significado de esta técnica, procederemos por una doble vía:

1. En la primera vía, comenzaremos por determinar el *embedding* asociado a la función *kernel* elegida y calcularemos de forma explícita las imágenes de los elementos originales por dicho *embedding*. Seguidamente centraremos los datos en el *feature space* y calcularemos la matriz de covarianzas en dicho *feature space*. Finalmente, la descomposición espectral de dicha matriz nos proporcionará sus valores y vectores propios y, a partir de ellos, podremos calcular las coordenadas de los individuos en los componentes principales (que no serán lineales). En pocas palabras, esta primera vía supone efectuar un *kernel PCA* sin recurrir al *kernel trick*. Conviene insistir en que esta vía no es en absoluto recomendable y se emprende únicamente a efectos ilustrativos.

2. En la segunda vía, dejaremos de lado el *embedding* y nos limitaremos a calcular la matriz *kernel* para los datos del ejemplo de ilustración. Antes de diagonalizar dicha matriz será necesario someterla a ciertas transformaciones para garantizar que los elementos están centrados en el *feature space*. Una vez transformada, la matriz *kernel* será diagonalizada y sus valores y vectores propios nos permitirán calcular las coordenadas de los individuos en los componentes principales. Esta segunda vía supone realizar un *kernel PCA* apoyándonos (como es natural) en el *kernel trick*. Huelga decir que ésta es la vía recomendable.

Comencemos, pues, con la primera vía.

Por su sencillez, el *kernel* elegido para este ejemplo de ilustración es un *kernel polinómico de grado 2 y offset nulo*, es decir:

$$k(x,z)=\langle x,z\rangle^2$$

Dado que el espacio en el que se encuentran inicialmente los individuos es $\mathbb{R}^3$ resulta que:

$$k(x,z)=(x_1z_1+x_2z_2+x_3z_3)^2=x_1^2z_1^2+x_2^2z_2^2+x_3^2z_3^2+2x_1z_1x_2z_2+2x_1z_1x_3z_3+2x_2z_2x_3z_3$$

es decir

$$k(x,z)=\langle(x_1^2,x_2^2,x_3^2,\sqrt{2}x_1x_2,\sqrt{2}x_1x_3,\sqrt{2}x_2x_3),(z_1^2,z_2^2,z_3^2,\sqrt{2}z_1z_2,\sqrt{2}z_1z_3,\sqrt{2}z_2z_3)\rangle$$

y como

$$k(x,z)=\langle\phi(x),\phi(z)\rangle$$

resulta que el *embedding* $\phi:\mathbb{R}^3\to\mathbb{R}^6$ asociado al *kernel polinómico de grado 2 con offset nulo* es:

$$\phi(x)=\phi(x_1,x_2,x_3)=(x_1^2,x_2^2,x_3^2,\sqrt{2}x_1x_2,\sqrt{2}x_1x_3,\sqrt{2}x_2x_3)$$

Este *embedding* nos permite el cálculo de las coordenadas de las imágenes de los elementos en el *feature space*, que es el primer paso en la que hemos denominado primera vía (o *kernel PCA* sin *kernel trick*).

El código de R del anexo R07 realiza todos los cálculos necesarios.

Tras ejecutar este código con los datos del ejemplo de ilustración obtenemos los siguientes resultados:

La matriz que recoge las imágenes de los individuos en el *feature space* es:

```
> X_F
       f1    f2 f3         f4         f5          f6
[1,] 2.56  4.84  1  4.9780317 -2.2627417  -3.1112698
[2,] 0.36  0.64  0 -0.6788225  0.0000000   0.0000000
[3,] 0.36 14.44  4 -3.2244069  1.6970563 -10.7480231
[4,] 0.16  0.04  1 -0.1131371 -0.5656854   0.2828427
[5,] 5.76  4.84  4 -7.4670476  6.7882251  -6.2225397
```

Resulta evidente que esta matriz no está centrada. Tras centrarla obtenemos:

```
> X_F_C
        f1    f2 f3        f4         f5         f6
[1,]  0.72 -0.12 -1  6.279108 -3.3941125  0.8485281
[2,] -1.48 -4.32 -2  0.622254 -1.1313708  3.9597980
[3,] -1.48  9.48  2 -1.923330  0.5656854 -6.7882251
[4,] -1.68 -4.92 -1  1.187939 -1.6970563  4.2426407
[5,]  3.92 -0.12  2 -6.165971  5.6568542 -2.2627417
```

La matriz de covarianzas (sin dividirla entre l) en el *feature space* es:

```
> CF
           f1          f2        f3        f4        f5         f6
f1  23.08800    0.07200   8.80000 -19.71979  23.41938  -11.20057
f2   0.07200  132.76800  32.40000 -26.77955  18.32821 -102.16279
f3   8.80000   32.40000  14.00000 -24.89016  19.79899  -31.11270
f4 -19.71979  -26.77955 -24.89016  82.94400 -60.00000   39.84000
f5  23.41938   18.32821  19.79899 -60.00000  48.00000  -31.20000
f6 -11.20057 -102.16279 -31.11270  39.84000 -31.20000   85.60000
```

y su descomposición espectral nos proporciona los siguientes vectores propios:

```
> W_CF
             [,1]        [,2]        [,3]        [,4]      [,5]        [,6]
[1,] -0.09419524 -0.26380411  0.84169707 -0.13393442 0.0000000  0.44174909
[2,] -0.64213674  0.49201298 -0.07695471 -0.22281247 0.4840814  0.23596904
[3,] -0.22143720 -0.07095825  0.09454979  0.94263902 0.2194647  0.01605465
[4,]  0.36602036  0.63336729  0.48373323  0.05490197 0.1627063 -0.44876449
[5,] -0.27725691 -0.50324683  0.13254264 -0.19827392 0.4012399 -0.67230771
[6,]  0.56472200 -0.16995312 -0.15848964 -0.03900108 0.7280340  0.30908151
```

cuyos respectivos valores propios son:

```
> LAMBDA_CF
[1]  2.569758e+02  1.101262e+02  1.853365e+01  7.643286e-01  1.238715e-14 -1.124508e-15
```

En consecuencia, las coordenadas (proyecciones) de los individuos sobre estas direcciones del *feature space* son:

```
> F_CF
           [,1]      [,2]      [,3]        [,4]          [,5]          [,6]
[1,]   3.949178  5.362826  2.973772 -0.02772839  1.103880e-15 -1.666365e-15
[2,]   6.133937 -1.302655 -1.578904 -0.62045730 -2.066923e-15  5.725238e-15
[3,] -11.085200  4.563623 -1.565681  0.01823156 -1.127570e-16 -1.259409e-15
[4,]   6.840241 -1.021165 -1.452685  0.61484282  2.806132e-15 -1.428220e-15
[5,]  -5.838156 -7.602629  1.623497  0.01511132 -1.498367e-15 -1.397482e-15
```

A efectos de comparaciones posteriores, vamos a calcular también los productos escalares entre las imágenes de los elementos. Lo conseguimos con el siguiente código de R, que acompañamos de su resultado:

```
> KF<-(X_F_C)%*%t(X_F_C)
> KF
       [,1]   [,2]   [,3]   [,4]   [,5]
[1,]  53.20  12.56 -23.96  17.20 -59.00
[2,]  12.56  42.20 -71.48  45.20 -28.48
[3,] -23.96 -71.48 146.16 -78.20  27.48
[4,]  17.20  45.20 -78.20  50.32 -34.52
[5,] -59.00 -28.48  27.48 -34.52  94.52
```

Finalizada la primera vía, comencemos con la segunda para, posteriormente, comparar los resultados.

El primer paso en el procedimiento del *kernel PCA* que se apoya en el *kernel trick* consiste en el cálculo de la matriz *kernel* asociada a los datos del ejemplo y a la función *kernel* elegida. A continuación hay que transformar la matriz *kernel* para asegurar que los datos se encuentran centrados en el *feature space*. Como vimos en anteriores apartados, la transformación que debe efectuarse es:

$$\hat{K} = K - \frac{1}{l} K\, j\, j^T - \frac{1}{l}\, j\, j^T K + \frac{1}{l^2}\, j^T K\, j\, j^T$$

donde j es un vector de unos.

Finalmente, la descomposición espectral de la matriz $\hat{K}$ proporciona los valores y vectores propios que permiten calcular las coordenadas de los individuos en los componentes principales.

El código de R del anexo R08 realiza los cálculos precisos.

La matriz *kernel* inicial, resultado de evaluar la función *kernel* elegida (polinómica de grado dos y *offset* nulo) es:

```
> KMI
      [,1]  [,2]   [,3] [,4]   [,5]
[1,] 70.56  0.64  88.36 1.44   9.00
[2,]  0.64  1.00  11.56 0.16  10.24
[3,] 88.36 11.56 353.44 1.00 190.44
[4,]  1.44  0.16   1.00 1.44   0.36
[5,]  9.00 10.24 190.44 0.36 213.16
```

Dado que todos los elementos de la matriz *kernel* son positivos resulta evidente que ésta no puede ser la matriz *kernel* asociada a un conjunto de datos centrado[20].

Una vez sometida a la transformación aludida anteriormente, la matriz *kernel* definitiva resulta ser:

```
> KM
       [,1]   [,2]   [,3]   [,4]   [,5]
[1,]  53.20  12.56 -23.96  17.20 -59.00
[2,]  12.56  42.20 -71.48  45.20 -28.48
[3,] -23.96 -71.48 146.16 -78.20  27.48
[4,]  17.20  45.20 -78.20  50.32 -34.52
[5,] -59.00 -28.48  27.48 -34.52  94.52
```

En este caso, el promedio de los elementos de la matriz es nulo, por lo que podemos asegurar que los datos están centrados en el *feature space*.

Como no podía ser de otra manera, la igualdad

$$K = [\phi(X)][\phi(X)]^T$$

hace que la matriz *kernel* que acabamos de calcular coincida con la matriz de productos escalares entre los elementos transformados por el *embedding* que calculamos al final de la llamada "primera vía" (matriz a la que denominamos KF en nuestro código de R).

Finalmente, diagonalizando esta matriz *kernel* encontramos sus vectores y valores propios, que resultan ser:

```
> LAMBDA_KM
[1]  2.569758e+02  1.101262e+02  1.853365e+01  7.643286e-01 -1.555236e-13
> W_KM
           [,1]        [,2]       [,3]        [,4]       [,5]
[1,] -0.2463546 -0.51103232  0.6907601  0.03171646 -0.4472136
[2,] -0.3826425  0.12413213 -0.3667543  0.70969518 -0.4472136
[3,]  0.6915083 -0.43487496 -0.3636828 -0.02085373 -0.4472136
[4,] -0.4267026  0.09730842 -0.3374356 -0.70327319 -0.4472136
[5,]  0.3641913  0.72446672  0.3771126 -0.01728472 -0.4472136
```

Nótese la coincidencia entre los valores propios calculados por ambas vías.

En cualquier caso, la matriz *kernel* y su descomposición espectral nos permiten determinar las coordenadas de los individuos sobre los componentes principales.

```
> F_KM
          [,1]      [,2]      [,3]        [,4] [,5]
[1,] -3.949178 -5.362826  2.973772  0.02772839  NaN
[2,] -6.133937  1.302655 -1.578904  0.62045730  NaN
[3,] 11.085200 -4.563623 -1.565681 -0.01823156  NaN
[4,] -6.840241  1.021165 -1.452685 -0.61484282  NaN
[5,]  5.838156  7.602629  1.623497 -0.01511132  NaN
```

20 Recuérdese que el cuadrado de la norma del centro de masas de los elementos de $\phi(S)$ es igual al promedio de los elementos de la matriz *kernel*.

que coinciden (una vez más, salvo el signo) con las calculadas mediante la vía en la que hemos establecido de forma explícita el *embedding*. Nótese que en ambas matrices de coordenadas sólo tienen sentido las cuatro primeras columnas ya que sólo hay cuatro valores propios no nulos.

Sirva este pequeño ejemplo de ilustración para, por una parte, poner de manifiesto la equivalencia entre el procedimiento basado en el cálculo explícito del *embedding* y el basado en la utilización exclusiva de la matriz *kernel* convenientemente transformada y, por otra, resaltar la superioridad en cuanto a eficiencia computacional de la segunda vía respecto a la primera: incluso en este caso tan sencillo en el que la dimensión del *feature space* es 6 resulta más adecuado el segundo enfoque; ¿qué decir en *feature spaces* de dimensión más elevada (que puede llegar a ser infinita)?

Con la resolución de este ejemplo de ilustración del *kernel PCA* se pone punto final a la presentación de los fundamentos teóricos de la incorporación de la metodología *kernel* a algunas técnicas seleccionadas de análisis supervisado (*support vector machines*) y aprendizaje no supervisado (análisis de componentes principales). El resto del trabajo se dedicará a proponer ciertos recursos de *software* para el tratamiento de las funciones *kernel* así como a la resolución de un par de ejemplos de cada una de las técnicas expuestas.

6. *Software* para el tratamiento de funciones *kernel* en *support vector machines* y en análisis de componentes principales

Entre las diversas opciones de *software* que existen actualmente para el tratamiento de las funciones *kernel* en el ámbito de las *support vector machines* y del análisis de componentes principales, se ha optado por los *packages* `e1071` y `kernlab` del entorno R.

El entorno de programación R

R es un lenguaje y un entorno de programación para el cálculo estadístico y para la generación de gráficos. Es un proyecto GNU[21] similar al lenguaje S, que fue inicialmente desarrollado en los Laboratorios Bell por John Chambers. R es una implementación alternativa de S.

R proporciona al usuario una amplia oferta de técnicas estadísticas y gráficas y es fácilmente extensible mediante la incorporación de los llamados *packages*.

En los últimos años, R se ha convertido en la *lingua franca* de la comunidad estadística y, virtualmente, cualquier nuevo desarrollo en este campo se ve materializado en la publicación de un nuevo *package*, que aumenta las capacidades básicas del entorno.

R está disponible como *Software Libre* bajo los términos de una *GNU General Public License* (Licencia Pública General de GNU: http://www.gnu.org/copyleft/gpl.html) de la *Free Software Foundation*. Existen versiones de R para UNIX, Windows y MacOS.

En la actualidad, R se encuentra en su versión 2.11.1, que fue liberada en mayo de 2010.

LIBSVM: una librería en C++ para el tratamiento de las support vector machines

LIBSVM es una biblioteca de programación (*library*) desarrollada en C++ y destinada a la aplicación de las *support vector machines*. Inicialmente desarrollada por Chih-Chung Chang y Chih-Jen Lin en 2001, se encuentra actualmente (agosto de 2010) en su versión 2.91, que fue liberada en abril de 2010.

LIBSVM proporciona al usuario diferentes herramientas para la clasificación y la regresión basadas en *support vector machines* e incluye utilidades que permiten la clasificación en múltiples clases.

21 GNU es un acrónimo recursivo introducido por Richard Stallman que significa “GNU no es Unix” (*GNU is Not Unix*)

`e1071`: interfaz entre LIBSVM y R

`e1071` es uno de los *packages* que desarrollan las capacidades básicas de R. Es un interfaz que permite utilizar desde R la biblioteca LIBSVM. Fue desarrollado por Evgenia Dimitriadou, Kurt Hornik, Friedrich Leisch, David Meyer y Andreas Weingessel en la *Technische Universität* de Viena y se encuentra, en la actualidad, en su versión 1.5.24 (de abril de 2010).

`kernlab`: un package de R dedicado a los métodos kernel

`kernlab` es otro *package* del entorno R, que está dedicado a la aplicación de métodos de aprendizaje estadístico basados en funciones *kernel*. Una de las particularidades de `kernlab` es que hace uso del nuevo modelo de datos desarrollado en R: el modelo S4.

`kernlab` proporciona al usuario diversas herramientas para el tratamiento de las funciones *kernel* en el ámbito de las *support vector machines* así como en el análisis de componentes principales.

El *package* `kernlab` incluye, entre sus utilidades, un optimizador cuadrático que es, precisamente, el que se ha empleado en la resolución de los ejercicios de ilustración.

En la actualidad, `kernlab` se encuentra en la versión 0.9.10 (abril de 2010) y sus autores son Alexandros Karatzoglou, Alex Smola y Kurt Hornik (autor, también de `e1071`) de la *Technische Universität* de Viena.

Una vez presentado el *software* elegido, en los siguientes apartados se procede a la aplicación de dicho *software* al análisis de distintas bases de datos.

Comenzaremos por las *support vector machines* y realizaremos las siguientes aplicaciones:

- Aplicación del *package* `e1071` al ejemplo de ilustración
- Aplicación del *package* `kernlab` al ejemplo de ilustración
- Aplicación del *package* `e1071` a una base de datos sobre variables de competitividad en las regiones europeas (NUTS2)
- Aplicación del *package* `kernlab` a la misma base de datos sobre variables de competitividad en regiones europeas

Seguidamente, afrontaremos el tratamiento del *análisis de componentes principales* mediante funciones *kernel*. Emplearemos para este particular el *package* `kernlab` y llevaremos a cabo los siguientes análisis:

- Aplicación del *package* `kernlab` a una base de datos artificial
- Aplicación del *package* `kernlab` al análisis de la base de datos sobre variables de competitividad en regiones europeas anteriormente aludida

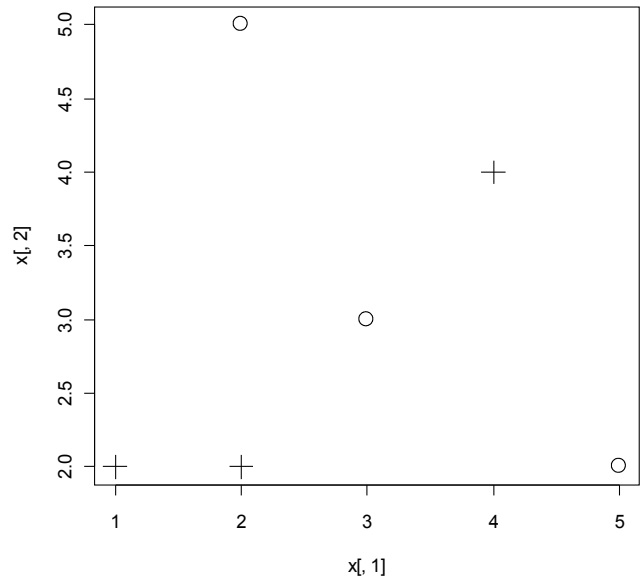
7. Dos aplicaciones de las *support vector machines*

Aplicación del package `e1071` al ejemplo de ilustración de support vector machines

Recordemos que el ejemplo de ilustración empleado en el caso de las *support vector machines* se refería al siguiente conjunto de datos no separable:

$$D=\{[(1,2)^T,+1],[(2,2)^T,+1],[(4,4)^T,+1],[(2,5)^T,-1],[(5,2)^T,-1],[(3,3)^T,-1]\}$$

cuya representación gráfica en el plano $\mathbb{R}^2$ es:



Recuérdese que los elementos de la clase +1 se denotan con una cruz mientras que los elementos de la clase -1 se representan con un círculo.

Clasificador de margen máximo *soft* con *kernel* lineal

Naturalmente, por tratarse de dos clases no separables, el problema de determinación del clasificador de margen máximo *hard* carece de sentido, por lo que, en el ejemplo de ilustración nos centramos en determinar el clasificador de margen máximo *soft* con parámetro de coste $C=5$.

A continuación procedemos a resolver este ejemplo con el *package* `e1071`.

Los datos del ejemplo se recogen en el archivo *DatosEjemploIlustracion.csv* cuyo contenido es el siguiente:

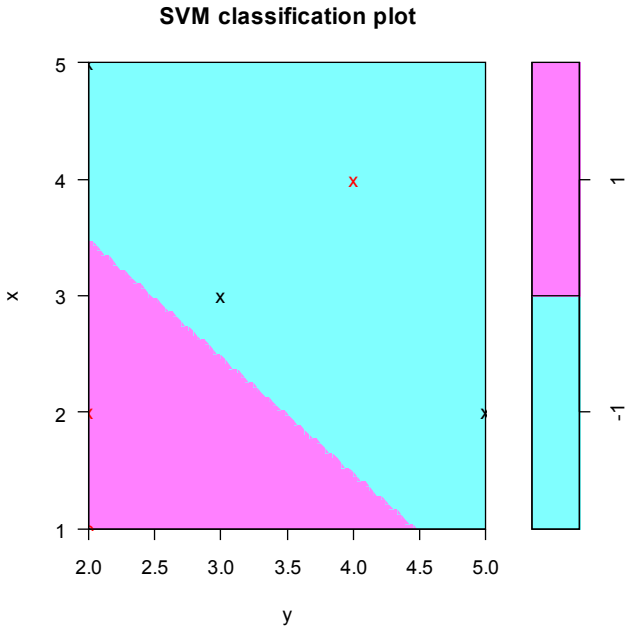
```
x;y;clase
1;2;1
2;2;1
4;4;1
2;5;-1
5;2;-1
3;3;-1
```

El código de R empleado para el ejemplo de ilustración se recoge en el anexo R09.

- En primer lugar, se invoca desde R el *package* `e1071` con la función `library()`.
- A continuación, con la función `read.csv()` se lee el archivo de datos y, seguidamente, se modifica la clase de su tercera columna (que se convierte en un factor).
- La función `svm()` permite la determinación del clasificador de margen máximo *soft* con un parámetro de coste $C=5$. Mención especial merece el argumento `scale=FALSE` que se incluye en dicha función. El motivo para asignar tal valor al argumento es que la función `svm()` procede (por defecto) a una tipificación de las variables (ya que esta operación proporciona habitualmente unos mejores resultados). En este caso, sin embargo, no deseamos que los datos sean tipificados y, por ese motivo, establecemos el valor `FALSE` para el argumento `scale`.
- Los resultados se obtienen accediendo de diferentes formas al objeto `svm1` creado con la función `svm()`:
  - El gráfico de clasificación se consigue con la función `plot()`
  - Los índices y coordenadas de los *support vectors* se obtienen mediante el comando `svm1$SV`
  - Los coeficientes α_i se consiguen mediante el comando `svm1$coef` aunque los resultados que proporciona `e1071` aparecen multiplicados por su etiqueta.
- La función `predict()` asigna a cada uno de los elementos de la muestra de aprendizaje la clase que le corresponde según la *support vector machine*.
- Finalmente, la función `table()` cruza la clasificación real con la clasificación de acuerdo a la *support vector machine*.

Los resultados son los siguientes:

```
> plot(svm1,datos.svm,grid=100)
```




```
> svm1$SV
  x y.1
2 2   2
3 4   4
4 2   5
5 5   2
6 3   3

> svm1$coefs
          [,1]
[1,]  3.777778
[2,]  5.000000
[3,] -1.888889
[4,] -1.888889
[5,] -5.000000

> pred<-predict(svm1,datos.svm[,1:2])
> table(pred,datos.svm[,3])

pred -1 1
  -1  3 1
  1   0 2
```

El gráfico muestra las dos zonas en las que la superficie óptima de decisión divide al plano $\mathbb{R}^2$ aunque no indica dónde se encuentran los *supporting hyperplanes*. Los *support vectors* se denotan con una cruz mientras que los elementos que no son *support vectors* aparecen representados con un círculo. Para representar cada una de las dos clases, se emplea un código de color: rojo para los elementos de la clase +1 y negro para los de la clase -1. En relación con el gráfico cabe hacer algunos comentarios:

- Los ejes de abscisas y ordenadas aparecen intercambiados respecto a lo que suele ser habitual. No parece que esta forma de proceder tenga sentido.

- Los límites de ambos ejes son excesivamente reducidos, lo que no permite apreciar correctamente dónde se encuentran ubicados algunos de los elementos. Por ejemplo, apenas es posible apreciar si el elemento de la clase +1 situado en las coordenadas (x=1,y=2) es o

no un *support vector*.

- Se echa de menos la representación de los *supporting hyperplanes* de cada una de las clases.

El resultado del comando `svm1$SV` muestra que los *support vectors* son los elementos 2, 3, 4, 5 y 6 -es decir, todos excepto el elemento de la clase +1 situado en las coordenadas (x=1,y=2)-. Nótese que el *package* `e1071` considera que los elementos 3 y 6 son *support vectors* (aunque no se encuentran situados sobre su *supporting hyperplane* sino mal posicionados con respecto a él). Esto es porque sus coeficientes asociados (proporcionados por el comando `svm1$coef`) son positivos. Sabemos, sin embargo, por la resolución manual del ejemplo, que estos vectores se encuentran situados en el lado incorrecto de su *supporting hyperplane* y que, en consecuencia, no deben ser considerados como *support vectors*.

A continuación, el comando `svm1$coef` proporciona los coeficientes α_i asociados a cada uno de los *support vectors* multiplicados por la etiqueta de su clase (los elementos que no son *support vectors* tienen coeficiente α_i nulo). Nótese la coincidencia con los resultados obtenidos en la resolución manual del ejemplo de ilustración.

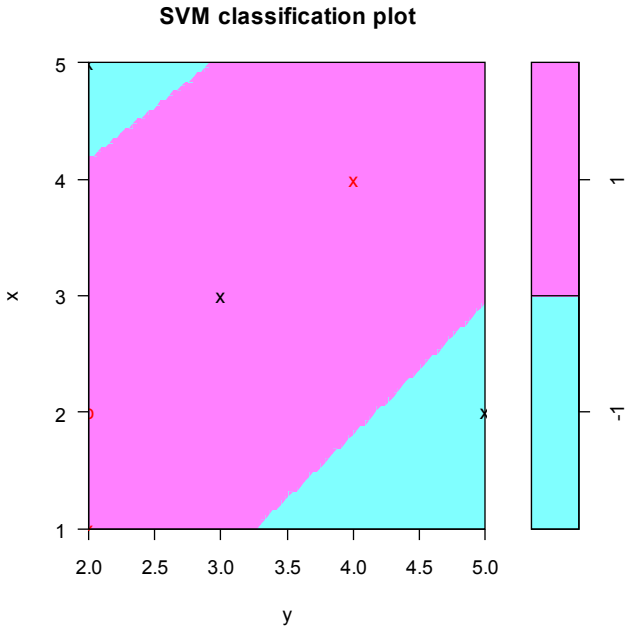
Finalmente, observamos que el clasificador óptimo encontrado comete un error en la muestra de aprendizaje ya que clasifica mal al elemento ubicado en las coordenadas (x=4, y=4).

Clasificador de margen máximo *soft* con kernel polinómico

Tras resolver el problema de encontrar el clasificador de margen máximo *soft* con parámetro de coste $C=5$ afrontamos, con el mismo ejemplo de ilustración, idéntica tarea pero considerando no el producto escalar habitual sino un *kernel polinómico homogéneo de grado 2*. Este problema puede resolverse con el *package* `e1071` mediante el código de R recogido en el anexo R10.

La única diferencia respecto al código anterior es la que se refiere a los argumentos de la función `svm()` que, en este caso, recogen las peculiaridades del *kernel polinómico homogéneo de grado 2*. Se incluye, de nuevo, el argumento `scale=FALSE` para evitar la tipificación que, por defecto, efectúa la función `svm()`.

Los resultados son los que se proporcionan a continuación:




```
> svm2$SV

  x y.1
1 1   2
3 4   4
4 2   5
5 5   2
6 3   3

> svm2$coefs
             [,1]
[1,]  2.653949944
[2,]  2.742752051
[3,] -0.387919100
[4,] -0.008782896
[5,] -5.000000000

> pred<-predict(svm2,datos.svm[,1:2])
> table(pred,datos.svm[,3])

pred -1 1
  -1  2 0
  1   1 3
```

Los resultados coinciden, naturalmente, con los obtenidos en la resolución manual:

- Los *support vectors* son los elementos 1 (x=1,y=2), 3 (x=4,y=4), 4 (x=2,y=5) y 5 (x=5,y=2), aunque `e1071` considera también como *support vector* al elemento 6 (x=3,y=3), a pesar de que se encuentra en el lado incorrecto de su *supporting hyperplane*.

- Los valores α_i asociados a los *support vectors* son todos positivos[22] (recuérdese que

22 Nótese que el comando `svm2$coef` proporciona sólo los coeficientes no nulos, por lo que los índices que se emplean en el resultado de este comando no se refieren al conjunto de individuos inicial sino al conjunto de *support vectors*. En concreto, el índice 5, que lleva asociado un coeficiente de -5 (lo que indica que se trata de un elemento

`e1071` proporciona los resultados de los coeficientes multiplicados por la etiqueta de la clase). El elemento que se encuentra mal situado respecto a su *supporting hyperplane,* ubicado en las coordenadas (x=3, y=3), presenta un valor α_i igual al parámetro de coste.

- El clasificador óptimo encontrado clasifica de forma equivocada un elemento de la clase -1. Se trata, en concreto del elemento cuyas coordenadas son (x=3,y=3).

Aplicación del package `kernlab` al ejemplo de ilustración

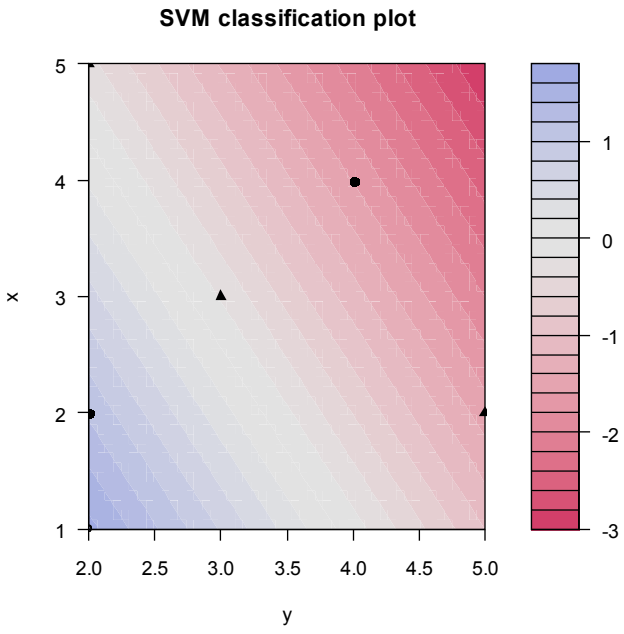
A continuación resolveremos los problemas de determinación del clasificador de margen máximo *soft* con parámetro de coste $C=5$ y *kernels* respectivos lineal y polinómico homogéneo de grado 2 pero, en esta ocasión, con el *package* `kernlab`.

Clasificador de margen máximo *soft* con kernel lineal

El código de R empleado para resolver el ejemplo de ilustración para el caso lineal con el *package* `kernlab` se encuentra en el anexo R11.

Los resultados son los siguientes:

El gráfico de clasificación es el que se muestra a continuación:



de la clase -1 mal ubicado respecto a su *supporting hyperplane*) se refiere al quinto *support vector*, es decir, al sexto elemento del conjunto inicial: el elemento de coordenadas (x=3, y=3).

Los coeficientes α_i correspondientes a los *support vectors* son:

```
> alpha(svp1)
[[1]]
[1] 3.777778 5.000000 1.888889 1.888889 5.000000
```

Los *support vectors* (incluidos los que se encuentran mal situados respecto a su *supporting hyperplane*) son los elementos 2, 3, 4, 5 y 6

```
> alphaindex(svp1)
[[1]]
[1] 2 3 4 5 6
```

Los coeficientes α_i multiplicados por la etiqueta de la clase son:

```
> coef(svp1)
[[1]]
[1]  3.777778  5.000000 -1.888889 -1.888889 -5.000000
```

El valor de b^* es:

```
> b(svp1)
[1] -3.666667
```

El número de *support vectors* es:

```
> nSV(svp1)
[1] 5
```

El valor de la función objetivo en el problema de optimización es:

```
> obj(svp1)
[1] -17.11111
```

La tasa de error en la muestra de aprendizaje es:

```
> error(svp1)
[1] 0.1666667
```

Por último, el clasificador óptimo se equivoca al clasificar uno de los elementos de la muestra de aprendizaje:

```
> pred<-predict(svp1,as.matrix(datos.svm[,1:2]))
> table(pred,datos.svm[,3])
pred -1 1
  -1  3 1
  1   0 2
```

Como se puede apreciar, `kernlab` proporciona algunos resultados más que los que facilita

`e1071`.

En cuanto al gráfico de clasificación, éste no muestra dos zonas (como en el caso de `e1071`) sino que presenta un degradado en colores indicativo del valor de $w^{*T}x-b^{*}$. En concreto, la frontera de decisión corresponde a aquellos puntos asociados con el valor 0 y los hiperplanos soporte corresponden con los puntos asociados a los valores +1 y -1. `kernlab` representa los elementos de la clase +1 con un círculo y los de la clase -1 con un triángulo. Los elementos que constituyen el conjunto de *support vectors* llevan asociado un icono sólido mientras que los elementos que no son *support vectors* se representan con un icono hueco. Al respecto de este gráfico pueden realizarse idénticos comentarios a los que se formulaban para el caso del *package* `e1071`:

- Los ejes de abscisas y ordenadas aparecen intercambiados respecto a lo que suele ser habitual. No parece que esta forma de proceder tenga sentido.

- Los límites de ambos ejes son excesivamente reducidos, lo que no permite apreciar correctamente dónde se encuentran ubicados algunos de los elementos.

- Se echa de menos la representación de los *supporting hyperplanes* de cada una de las clases.

Adicionalmente, en el caso del gráfico de clasificación que proporciona el *package* `kernlab` cabe destacar que el degradado en colores que representa el valor de $w^{*T}x-b^{*}$ está incorrectamente elaborado. En concreto, para los datos del ejemplo de ilustración, el degradado de color debería "correr paralelo" a la recta que pasa por los puntos (x=5,y=2) y (x=2,y=5), que son los que determinan el *supporting hyperplane* asociado a la clase -1. Claramente se aprecia que esto no ocurre así en el gráfico proporcionado por `kernlab`.

Para tratar de resolver este problema y paliar las deficiencias que se han encontrado en los gráficos de clasificación proporcionados tanto por `e1071` como por `kernlab` puede emplearse el código de R del anexo R12.

De esta forma se consigue que:

- Los ejes de ordenadas y abscisas ocupen sus posiciones tradicionales

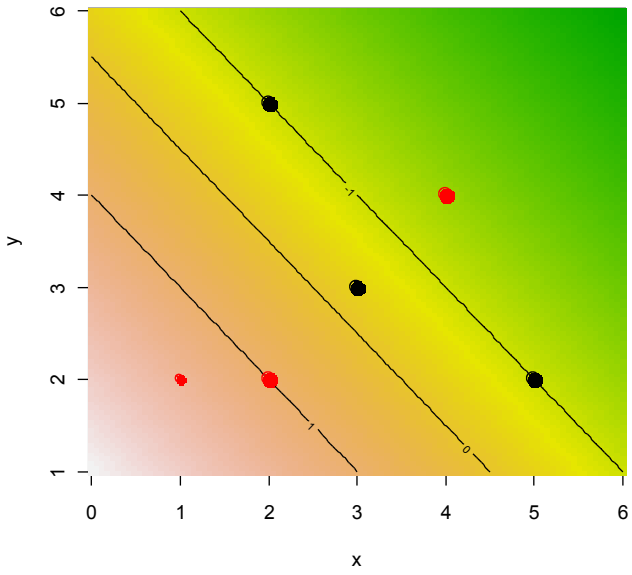
- Los puntos asociados a los elementos de la muestra se visualicen correctamente

- Se muestre un degradado en colores correctamente alineado

- Se represente claramente la frontera de decisión óptima

- Se representen claramente los hiperplanos soporte asociados a cada una de las dos clases

El resultado de este código de R es el siguiente gráfico:



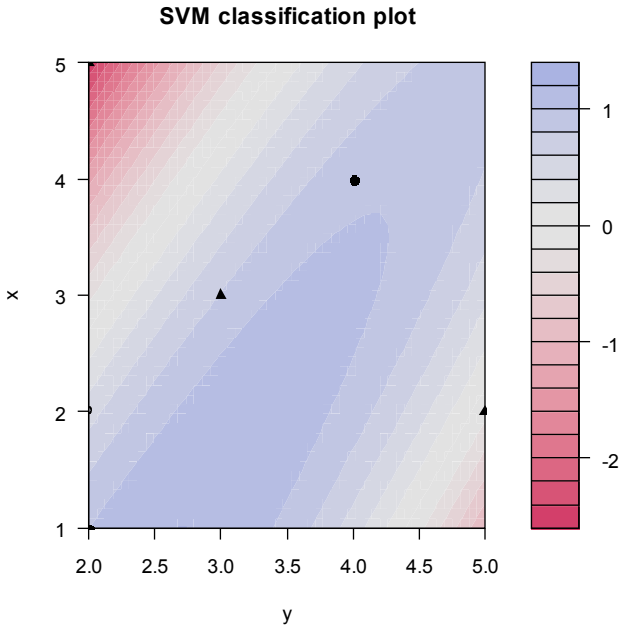
Los elementos de la clase +1 se señalan con un icono de color rojo y el color negro queda reservado a los elementos de la clase -1. Los *support vectors* se denotan con un icono de mayor tamaño e intensidad. Asimismo, el *supporting hyperplane* asociado a la cada una de las clases se identifica con su correspondiente etiqueta mientras que la frontera óptima de decisión es la línea recta que lleva asociada la etiqueta 0.

Clasificador de margen máximo *soft* con kernel polinómico

En cuanto a la aplicación a los datos del ejemplo de ilustración de un clasificador óptimo con parámetro de coste $C=5$ y *kernel* polinómico homogéneo de grado 2, el código de R es el que se encuentra en el anexo R13.

Los resultados son los que se presentan a continuación:

El gráfico de clasificación:



Los coeficientes α_i :

```
> alpha(svp2)
[[1]]
[1] 2.65395163 2.74275658 0.38792210 0.00878611 5.00000000
```

Los *support vectors* son los elementos 1, 3, 4, 5 y 6:

```
> alphaindex(svp2)
[[1]]
[1] 1 3 4 5 6
```

Los coeficientes α_i multiplicados por la etiqueta de la clase son:

```
> coef(svp2)
[[1]]
[1]  2.65395163  2.74275658 -0.38792210 -0.00878611 -5.00000000
```

El valor de b^* es:

```
> b(svp2)
[1] -1.266911
```

El número de *support vectors* es:

```
> nSV(svp2)
[1] 5
```

El valor de la función objetivo en el problema de optimización asociado al clasificador óptimo es:

```
> obj(svp2)
[1] -10.68840
```

La tasa de error de clasificación es:

```
> error(svp2)
[1] 0.1666667
```

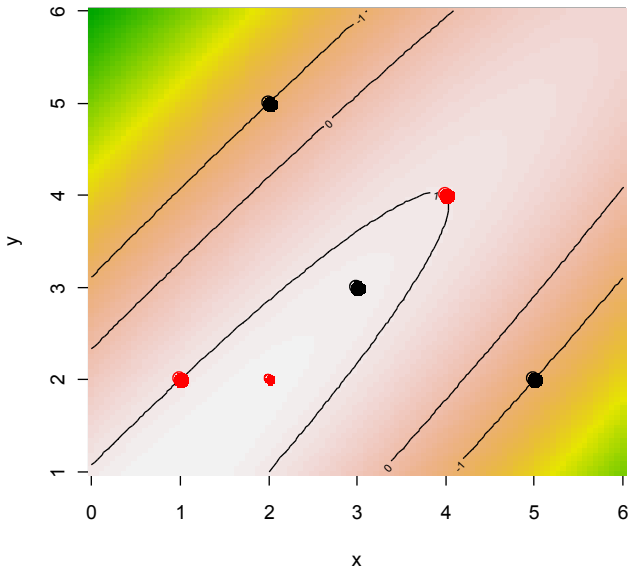
Finalmente, la matriz de confusión que cruza la verdadera clasificación de los elementos de la muestra con la clasificación prevista por la función de decisión es la siguiente:

```
> pred<-predict(svp2,as.matrix(datos.svm[,1:2]))
> table(pred,datos.svm[,3])

pred -1 1
  -1  2 0
  1   1 3
```

La observación del gráfico de clasificación que proporciona `kernlab` pone, una vez más, de manifiesto que el degradado en colores no está correctamente construido. Por ejemplo, los elementos situados en (x=1, y=2) y en (x=4, y=4) son *support vectors* con parámetro α_i menor que C de la clase +1 y deberían, por lo tanto, llevar asociado el valor +1 en la función que fija el degradado. Análogamente, el elemento cuyas coordenadas son x=2 e y=2 no es un *support vector* y pertenece a la clase +1. En consecuencia, este elemento debería estar en el lado correcto del hiperplano soporte asociado a la clase +1. Esto no es así en el gráfico.

El anexo R14 proporciona el código de R que permite elaborar un gráfico correcto.

El resultado se muestra a continuación:



Aplicación del package `e1071` a una base de datos sobre variables de competitividad en las regiones europeas (NUTS2)

Base de datos de regiones europeas y variables de competitividad

En este apartado se empleará una *support vector machine* sobre una base de datos de 188 regiones europeas (nivel NUTS2) en la que se recogen valores para un total de 25 variables relativas al ámbito del desempeño económico y tecnológico y, en general, relacionadas con la competitividad de dichas regiones. Esta base de datos fue elaborada por Navarro *et al.* (2009) y se utiliza con permiso de los autores.

Como tratamiento previo de los datos, se realiza un análisis de componentes principales tradicional (lineal) normado y ponderado con la población de cada una de las regiones. Se retienen las puntuaciones en los dos primeros componentes para las 188 regiones europeas. El porcentaje de la variabilidad total que se conserva en estos dos primeros componentes principales globalmente considerados es un 56,77%.

Respecto a la interpretación que cabe dar a cada uno de los dos componentes principales valga decir que:

- El primer componente principal, que recoge un 43,26% de la variabilidad total, es una variable que mide el desarrollo económico, tecnológico y social de la región considerada. Valores altos en este primer componente principal están asociados con regiones avanzadas desde el punto de vista económico, tecnológico y social.

- En cuanto al segundo componente principal, que aglutina un 13,51% de la variabilidad total, es una nueva variable que pone de manifiesto el carácter industrial (en oposición a un carácter agrícola o basado en servicios) de las regiones analizadas. Los valores altos en el

segundo componente principal están asociados con regiones en las que el peso relativo del sector industrial es muy elevado.

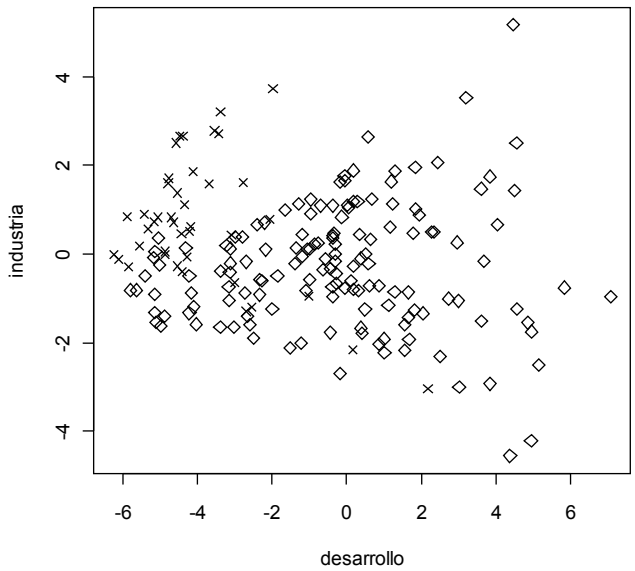
Adicionalmente, las 188 regiones han sido clasificadas en función de su pertenencia a un país de la UE-15 o de la UE-10. La etiqueta UE-15 o UE-10 viene determinada por el momento en el que el país al que se encuentra adscrita la región considerada entró a formar parte de la Unión Europea. En concreto:

- Son regiones de la UE-15 las que pertenecen a los siguientes países: Austria, Bélgica, Dinamarca, Finlandia, Francia, Alemania, Grecia, Irlanda, Italia, Luxemburgo, Holanda, Portugal, España, Suecia y Reino Unido. Los países de la UE-15 son los que formaban parte de la Unión Europea con anterioridad al 1 de mayo de 2004.

- Son regiones de la UE-10 las que pertenecen a los siguientes países: República Checa, Hungría, Polonia, Eslovaquia, Eslovenia, Letonia, Lituania, Estonia, Chipre y Malta. Estos países se incorporaron a la Unión Europea el 1 de mayo de 2004.

Una de las cuestiones que interesa a quienes estudian la realidad regional de la Unión Europea es la cohesión entre dichas regiones. Parece natural que las regiones de la UE-15, por un lado, y las de la UE-10, por otro, presenten diferentes características en cuanto a su estructura sectorial y su desarrollo económico, tecnológico y social. También parece razonable considerar que dichas diferencias deberían ir disminuyendo con el tiempo. Ese es, al menos, el objetivo de las políticas de cohesión que se han venido poniendo en marcha en el seno de la Unión Europea.

En el gráfico que se muestra a continuación se presentan los valores de las 188 regiones en los dos componentes principales (que han sido etiquetados con un rótulo ilustrativo). Las regiones pertenecientes a la UE-15 se representan con un rombo mientras que las regiones de la UE-10 llevan asociada una cruz.



La observación del gráfico pone de manifiesto que, aunque no de un modo determinista, parece existir una clara relación entre pertenecer a la UE-10 o a la UE-15 y el valor tomado en los índices de industria y de desarrollo (llamaremos así a partir de ahora a las puntuaciones que las regiones obtienen en los dos primeros componentes principales). En efecto:

- Las regiones de la UE-15 aparecen asociadas mayoritariamente a valores altos de desarrollo y las que presentan menores valores en dicho índice se encuentran situadas en valores cercanos a la media en el índice de industria.

- Las regiones de la UE-10, por el contrario, se encuentran mayoritariamente asociadas con valores bajos en el índice de desarrollo (salvo raras excepciones que corresponden a las regiones capital de los países de la UE-10) y también con valores altos del índice de industria.

A partir de esta base de datos se pretende entrenar una *support vector machine* para clasificar a las regiones europeas en regiones UE-15 o regiones UE-10 en función de los valores que tomen en los índices de desarrollo e industria. Dado que, como resulta evidente de la observación del gráfico anterior, las dos clases consideradas no son separables, carece de sentido tratar de ajustar un clasificador de margen máximo de tipo *hard*. En su lugar, consideraremos el entrenamiento de una *support vector machine* de tipo *soft* mediante la aplicación adicional de una función *kernel*.

En este caso, y en contraste con la forma de proceder en el caso del ejemplo de ilustración, vamos a considerar dos aspectos adicionales:

- El tamaño de la muestra es lo suficientemente amplio como para considerar la conveniencia de dividir la muestra global en una *muestra de aprendizaje*, que servirá para entrenar la *support vector machine*, y una *muestra test*, que se empleará con el fin de evaluar, *ex post*, la eficacia de la regla de decisión.

- El elemento clave de la aplicación de los métodos *kernel* a las *support vector machines* es la elección de un *kernel* adecuado, que permita que el algoritmo de optimización recoja toda la información necesaria acerca de los datos así como también acerca de las estructuras o patrones que cabe encontrar en la base. Siguiendo las recomendaciones de Chang y Lin (2010b), se seguirá un procedimiento de *tuning* de la *support vector machine*, eligiendo como familia de *kernels* a considerar, los *kernels* de tipo gaussiano. Emplearemos con este fin el *package* `e1071` y, en concreto, su función `tune.svm()`. El aludido procedimiento de *tuning* permite determinar cuáles son los valores más adecuados de los parámetros del *kernel* para la base de datos que se está considerando.

Creación de la base de datos para la *support vector machine*

Los datos del ejemplo se encuentran recogidos en el archivo *DatosDefinitivos.csv*, que contiene los datos de las 188 regiones en las 25 variables originales.

El código de R del anexo R15:

- Lee los datos desde este archivo

- Efectúa una transformación logarítmica sobre una variable con asimetría positiva.

- Realiza un análisis de componentes principales normado y ponderado (con el *package* FactoMineR) y recoge las puntuaciones de las 188 regiones en los dos primeros componentes principales, guardándolos en el `data.frame` de nombre `datos.svm`. La tercera columna de dicho `data.frame` recoge la clase a la que pertenecen las regiones: +1 para las regiones de la UE-15 y -1 para las regiones de la UE-10.

Tuning del kernel gaussiano

A partir de los datos recogidos en el `data.frame datos.svm` procederemos, siguiendo las recomendaciones de Chang y Lin (2010b), al *tuning* del *kernel* gaussiano. Los mencionados autores señalan la conveniencia de comenzar por un *kernel* de este tipo dada su mayor versatilidad.

El *package* `e1071` de R contiene la función `tune.svm()`, que permite el *tuning* de una *support vector machine*. Esta función devuelve como resultado un objeto de la clase `tune`, cuyo contenido es el conjunto de parámetros óptimo para la muestra analizada. Como medida del desempeño (*performance*) de cada una de las combinaciones de parámetros, la función `tune.svm()` emplea (en el caso de la clasificación) el error de clasificación.

El código del anexo R16 sirve para proceder al *tuning* del *kernel* gaussiano en el caso que nos ocupa. Se consideran todas las posibles combinaciones en las que el parámetro *gamma* se encuentra entre 0,25 y 4 y en las que el parámetro de coste se encuentra entre 1 y 16. Se emplea el método de muestreo `fix`, por el que en el proceso de *tuning* se considera una única vez la separación entre muestra de aprendizaje y muestra test. Este argumento puede tomar otros valores como por ejemplo `boot` (en cuyo caso se realiza un *bootstrapping*).

El resultado es el siguiente:

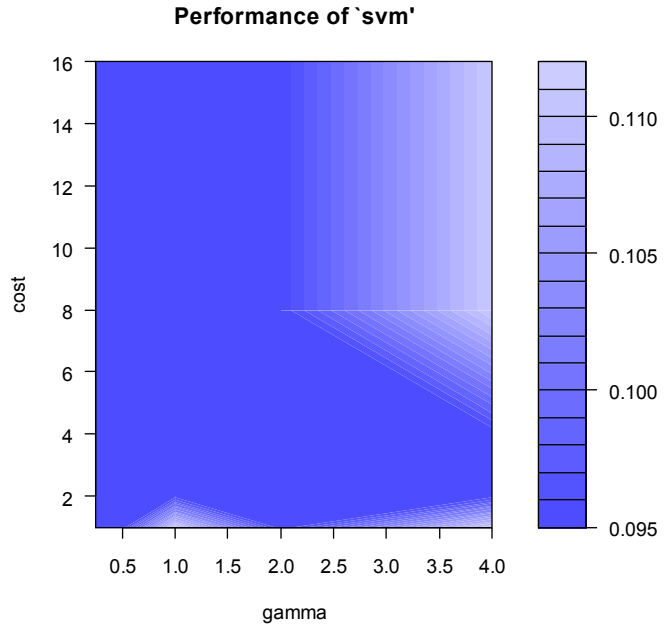
```
> obj

Parameter tuning of 'svm':

- sampling method: fixed training/validation set

- best parameters:
 gamma cost
  0.25    1

- best performance: 0.0952381
```

En el caso de utilizar un procedimiento de *bootstrapping*, el resultado obtenido para los parámetros es el mismo, aunque el rendimiento es mejor:

```
Parameter tuning of 'svm':

- sampling method: bootstrapping

- best parameters:
 gamma cost
  0.25    1

- best performance: 0.07763706
```

Adicionalmente, en el caso en el que se considere sólo un par de parámetros (como es el caso del *kernel* gaussiano) es posible obtener una representación gráfica del proceso de *tuning* mediante el

comando `plot(obj)`, cuyo resultado es el que se presenta a continuación:



Tanto empleando la técnica de muestreo fijo o por *bootstrapping*, el mejor rendimiento de la *support vector machine* se produce para la combinación de parámetros *gamma*=0,25 y $C=1$. El error de clasificación en el caso de muestreo fijo se sitúa en un 9,52% mientras que en el caso del *bootstrapping*, el desempeño mejora y se sitúa en una tasa de error del 7,76%. En consecuencia, los valores de los parámetros que emplearemos para el *kernel* radial serán *gamma*=0,25 y $C=1$.

Aplicación del *kernel* gaussiano elegido a la muestra de aprendizaje y a la muestra test

Una vez establecidos los parámetros más adecuados para el *kernel* gaussiano en el caso de la base de datos analizada, procedemos a dividir la muestra de 188 regiones en dos partes:

- 125 regiones compondrán la *muestra de entrenamiento* o *muestra de aprendizaje*. Esta muestra servirá para establecer la frontera óptima de decisión y, con ella, el criterio de decisión.

- 63 regiones conformarán la *muestra test*, cuyo objetivo es servir como piedra de toque para la *support vector machine* elaborada a partir de la muestra de aprendizaje.

El código de R del anexo R17:

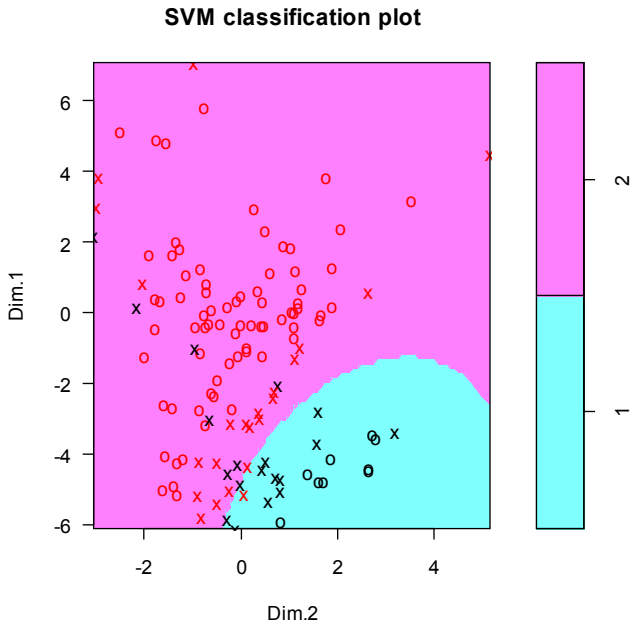
- Selecciona aleatoriamente 125 regiones y guarda sus datos en el `data.frame` de nombre `datostraining`.

- Los datos de las 63 regiones restantes se recogen en el `data.frame` de nombre `datostest`.

- A continuación se procede a entrenar una *support vector machine* a partir de los datos recogidos en la muestra de aprendizaje. Empleamos, para ello, el *package* `e1071`.

- Finalmente se aplica la *support vector machine* aprendida con la muestra de entrenamiento a los datos de la muestra test.

Los resultados básicos para la muestra de aprendizaje se presentan a continuación:




```
predtraining  1  2
           1 19  1
           2  9 96
```

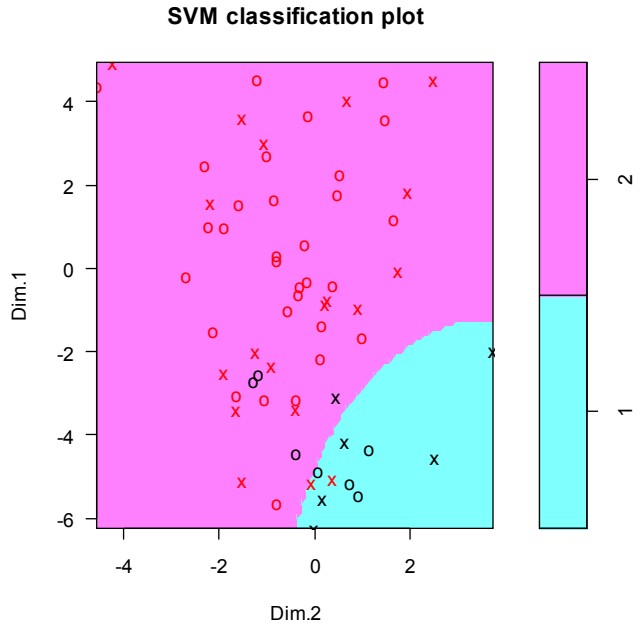
En definitiva, en la muestra de aprendizaje hay un total de 125 regiones, de las cuales 28 (el 22,4%) son de la UE-10 y 97 (el 77,6%) son de la UE-15. La *support vector machine* determinada consigue:

- Clasificar correctamente a 19 de las 28 regiones de la UE-10. Es decir, entre las regiones de la UE-10 el error de clasificación es de un 32,14%

- Clasificar correctamente a 96 de las 97 regiones de la UE-15. Entre las regiones de la UE-15, el error de clasificación es del 1,03%

- En resumen, la *support vector machine* clasifica correctamente a 115 de las 125 regiones que conforman la muestra de aprendizaje, por lo que el error de clasificación global es del 8%

Los resultados para la muestra test se presentan a continuación:




```
predtest  1  2
       1  9  2
       2  4 48
```

En el caso de la muestra test hay un total de 63 regiones, de las cuales 13 (el 20,6%) son de la UE-10 y 50 (el 79,4%) son de la UE-15. La *support vector machine* determinada a partir de la muestra de entrenamiento consigue:

- Clasificar correctamente a 9 de las 13 regiones de la UE-10. Es decir, entre las regiones de la UE-10 el error de clasificación es de un 30,77%

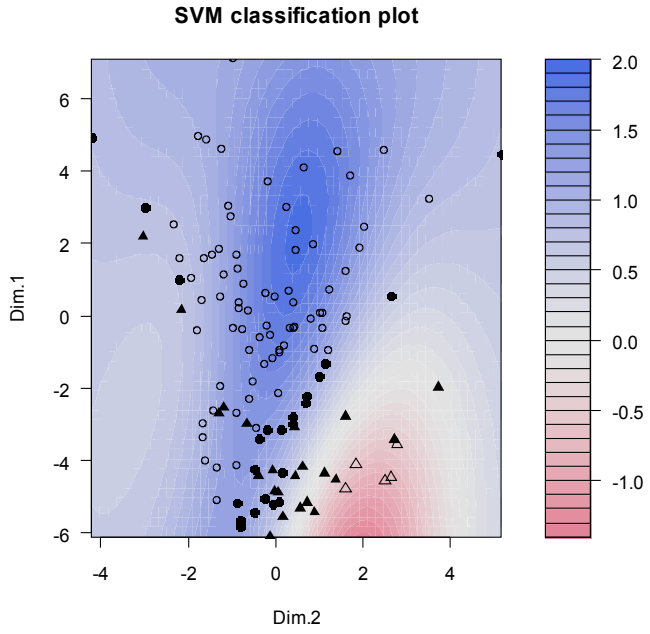
- Clasificar correctamente a 48 de las 50 regiones de la UE-15. Entre las regiones de la UE-15, el error de clasificación es del 4%

- En resumen, la *support vector machine* clasifica correctamente a 57 de las 63 regiones que conforman la muestra de aprendizaje, por lo que el error de clasificación global es del 9,52%, que coincide exactamente con la tasa de error que se obtuvo en el procedimiento de *tuning* del *kernel* radial.

Tratamiento de la base de datos de regiones europeas con el package `kernlab`

El procedimiento seguido con el *package* `e1071` puede reproducirse en gran medida con el *package* `kernlab` de R. Los comandos a emplear para obtener estos resultados se encuentran en el anexo R18.

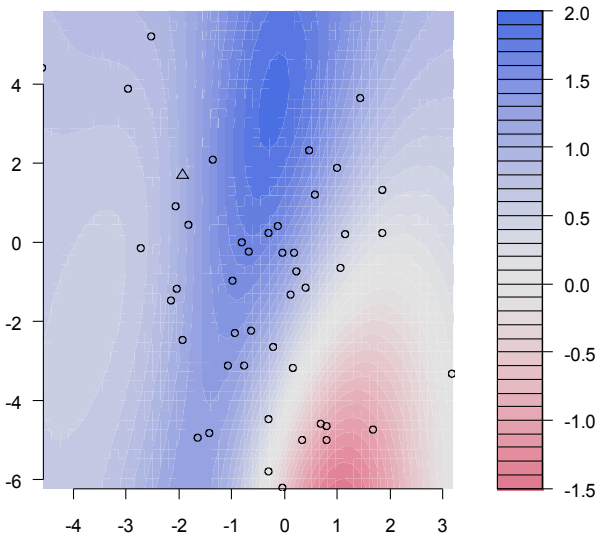
Los resultados son:



Los resultados para la muestra de aprendizaje son:

```
> predtraining<-predict(svp2,as.matrix(xtraining))
> table(predtraining,ytraining)
            ytraining
predtraining  1  2
           1 21  2
           2  9 93
```

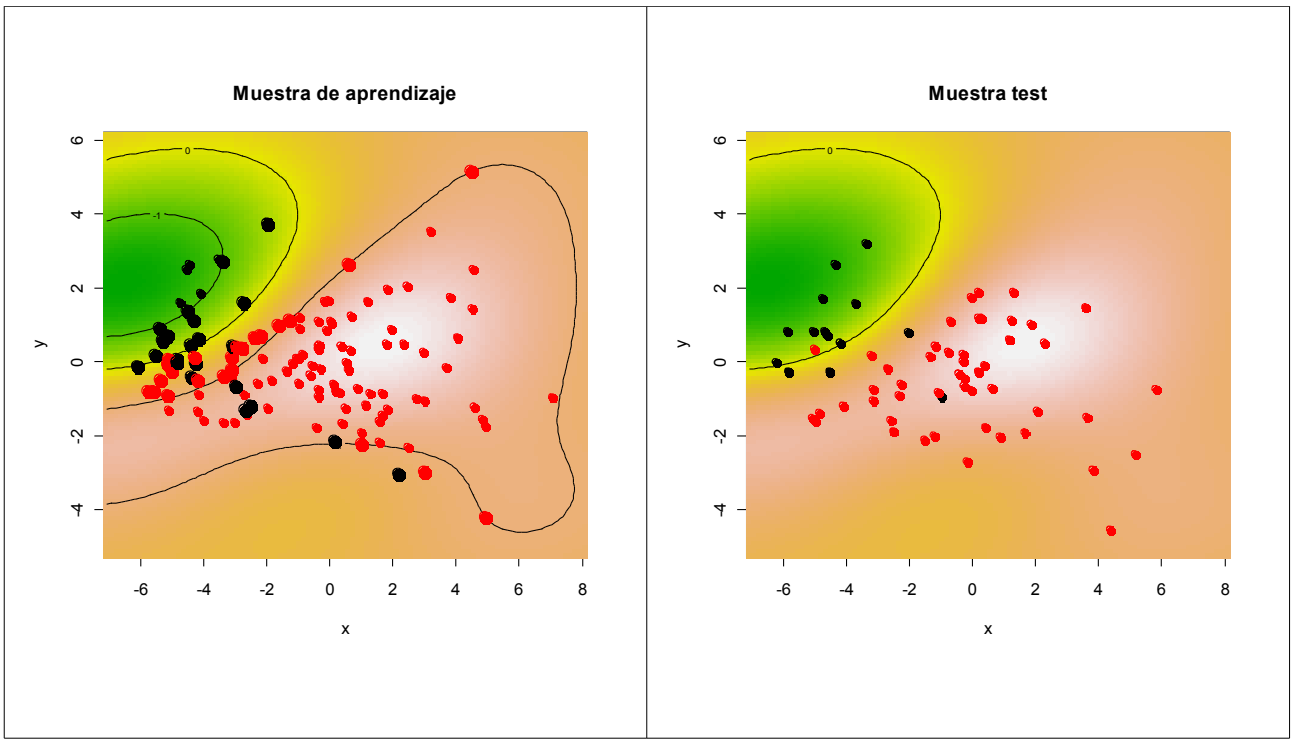
Los resultados para la muestra test son:




```
> predtest<-predict(svp2,as.matrix(xtest))
> table(predtest,ytest)
        ytest
predtest  1  2
       1  5  0
       2  6 52
```

Como se puede apreciar, los resultados difieren ligeramente de los obtenidos con el *package* `e1071`. Esta diferencia es completamente achacable al hecho de que se ha procedido a realizar una nueva selección de regiones para la muestra de aprendizaje y para la muestra test.

Como ya se ha comentado, los gráficos proporcionados tanto por `e1071` como por `kernlab` adolecen de ciertas deficiencias. Para superarlas puede utilizarse el código de R del anexo R19, cuyos resultados son los gráficos para la muestra de aprendizaje y para la muestra test que se presentan seguidamente:



En este tema hemos puesto en práctica la teoría expuesta acerca de las *support vector machines* empleando para ello dos *packages* de R: `kernlab` y `e1071`. Hemos trabajado con dos bases de datos muy diferentes en cuanto a su tamaño: la primera, la base de datos del ejemplo de ilustración; la segunda, una base de datos real sobre variables de competitividad en regiones europeas.

Cabe destacar que las representaciones gráficas de ambos *packages* adolecen de algunos defectos. Para tratar de superarlos se han propuesto nuevas representaciones gráficas y se ha proporcionado el código de R que permite obtenerlas.

El siguiente tema (el último antes del comentario final) es paralelo a éste y se dedica a la puesta en práctica, mediante su aplicación a un par de ejemplos y empleando el *package* `kernlab` de R, de la teoría del *kernel PCA*.

8. Dos ejemplos de aplicación del análisis de componentes principales basado en funciones *kernel*

Tras mostrar la aplicación de los *packages* `e1071` y `kernlab` al aprendizaje estadístico basado en funciones *kernel* en el caso de las *support vector machines* pasamos, a continuación, a aplicar nuevamente el *package* `kernlab` pero, en esta ocasión, en el ámbito del análisis de componentes principales.

Se tratarán dos casos:

- En primer lugar, se aplicará la técnica del *kernel PCA* (mediante el *package* `kernlab`) a una base de datos artificial creada *ad hoc*. Esta base de datos, de reducido tamaño, presenta la particularidad de que el conjunto de datos muestra tales valores en sus dos variables que una aplicación clásica del análisis de componentes principales no detecta ninguna *dirección principal de inercia*. Sin embargo, cuando se sustituye el producto escalar ordinario por una función *kernel* adecuada, el *kernel PCA* hace aflorar con suma facilidad la estructura que subyace a los datos.

- En segundo lugar, se aplicará idéntica técnica a un par de variables seleccionadas de la base de datos de regiones europeas que se empleó en el ejemplo de *support vector machines*.

Aplicación del package `kernlab` a una base de datos artificial

Los datos para este ejemplo artificial se encuentran recogidos en el archivo *ToyExampleData.csv*. Los comandos de R recogidos en el anexo R20 permiten leer los datos y mostrar las características principales del conjunto de datos:

```
> dim(datos)
[1] 300   3

> summary(datos)
       X                  Y            clase
 Min.   :-21.5400   Min.   :-21.5400   1:100
 1st Qu.: -4.8125   1st Qu.: -5.5075   2:100
 Median : -0.1350   Median :  0.0450   3:100
 Mean   :  0.1431   Mean   : -0.1905
 3rd Qu.:  6.4875   3rd Qu.:  4.1775
 Max.   : 21.8900   Max.   : 21.4100
```

El `data.frame` consta de 300 individuos clasificados (según la variable `clase`) en tres modalidades[23].

Comencemos por realizar un análisis de componentes principales clásico (es decir, empleando el producto escalar tradicional) sobre este conjunto de datos. Utilizaremos para este fin, al igual que hicimos en el ejemplo de *support vector machines*, el *package* FactoMineR. El comando para

23 Puede llamar la atención el hecho de que la muestra esté etiquetada y se pretenda aplicar una técnica de análisis no supervisado. El etiquetado de los elementos que componen la muestra responde a un interés ilustrativo que no es otro que el de facilitar la comparación de las representaciones de los individuos en el *input space* y en las direcciones principales de inercia del *feature space*.

efectuar un análisis de componentes principales no normado (es decir, empleando la matriz de covarianzas en lugar de la matriz de correlaciones) es el siguiente:

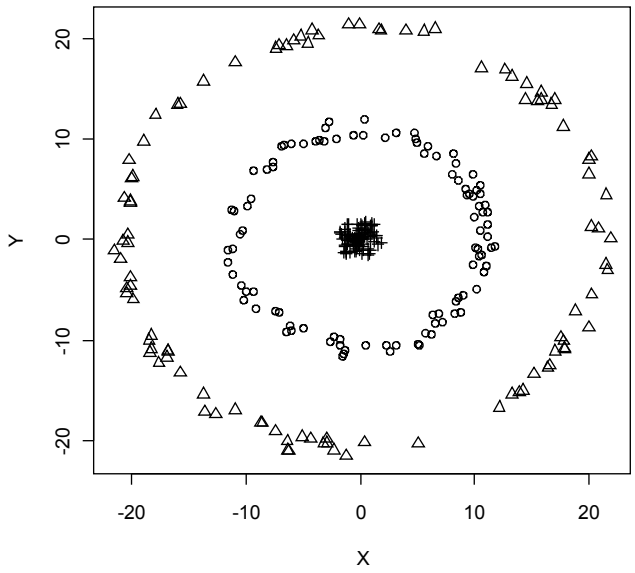
```
acp1<-PCA(datos[,1:2],scale.unit=FALSE)
```

El resultado de este análisis de componentes principales tradicional es el siguiente:

```
> acp1$eig
       eigenvalue percentage of variance cumulative percentage of variance
comp 1   102.0066              54.76517                          54.76517
comp 2    84.2552              45.23483                         100.00000
```

Es decir, hay un primer componente principal, que recoge un 54,77% de la inercia y un segundo componente que recoge el 45,23% de la variabilidad total de los datos respecto a su centro de masas. No obstante, la escasa diferencia entre los porcentajes de variabilidad recogidos en cada uno de los dos componentes principales permite afirmar que la dirección principal de alargamiento de la nube de individuos no está muy definida.

Una representación gráfica de los datos, distinguiendo su valor en la variable clase, puede ayudar a comprender mejor la estructura de la base que estamos analizando.



Resulta evidente que existe una estructura en los datos:

- Los 100 elementos de la clase 1 (representados con una cruz) se encuentran muy próximos al centro de masas
- Los 100 elementos de la clase 3 (representados por un triángulo) se encuentran muy alejados del centro de masas
- Los 100 elementos de la clase 2 (denotados por un círculo) se encuentran en una posición

intermedia.

Existe una estructura en los datos, sí, pero ésta no es lineal. Así, el análisis de componentes principales tradicional (basado en el producto escalar habitual) no es capaz de detectar esta estructura. Sin embargo, si sustituimos el producto escalar habitual por una función *kernel* adecuada quizás sea posible detectar dicha estructura.

Como ya se ha comentado con anterioridad, el elemento clave en el uso de funciones *kernel* para el aprendizaje estadístico es la elección de la función *kernel* adecuada a los datos y a las estructuras o patrones que cabe encontrar en ellos. En este caso se ha optado (tras un proceso iterativo de *ensayo y error*) por emplear un *kernel* gaussiano con parámetro gamma=0,00722, ya que es el que ha proporcionado unos mejores resultados.

La aplicación de la técnica del *kernel PCA* con el *package* `kernlab` se realiza mediante el siguiente comando de R:

```
kpca2<-kpca(~.,as.data.frame((datos[,1:2])),kernel="rbfdot",kpar=list(sigma=0.00722),features=2)
```

Como resultado de este código de R se obtiene:

- Los valores propios correspondientes a la matriz *kernel* (transformada convenientemente para que el centro de masas de los individuos en el *feature space* coincida con el origen). Para ello se emplea el comando `eig()`.

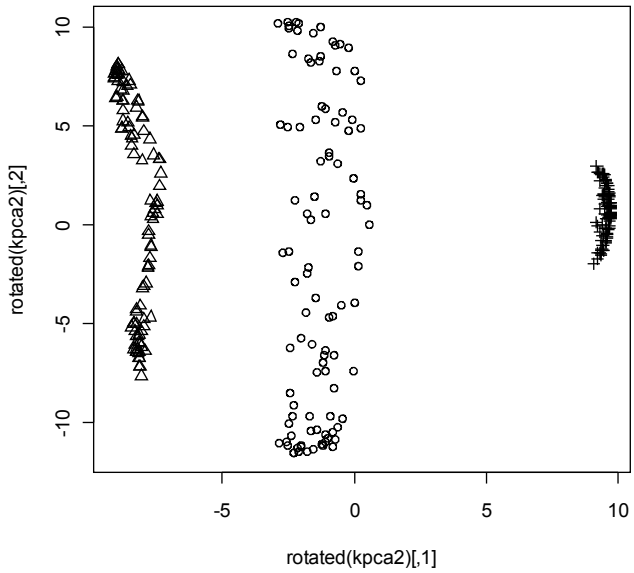
- Los vectores propios de la matriz *kernel* mediante el comando `pcv()`.

- Las coordenadas de los individuos en los componentes principales, que son proporcionales a los vectores propios de la matriz *kernel.* Para ello se utiliza el comando `rotated()`.

Mostramos, a continuación, los valores propios resultantes del *kernel PCA* acompañados por una representación gráfica[24] de las coordenadas de los individuos sobre los dos primeros componentes principales extraídos empleando el *kernel* gaussiano con gamma=0,00722:

```
> eig(kpca2)
   Comp.1    Comp.2
0.1802901 0.1046224
> plot(rotated(kpca2),pch=as.integer(datos$clase))
```

24 El etiquetado de los elementos que componen la muestra permite ver el efecto de sustituir el producto escalar habitual por un *kernel* gaussiano (véase la anterior nota al pie).



En cuanto a los valores propios cabe decir que el procedimiento de *kernel PCA* ha conseguido extraer un primer componente principal cuyo valor propio asociado es muy superior al segundo (0,18 frente a 0,10).

Asimismo, el gráfico anterior muestra que el primer componente principal ha tenido éxito en la detección de la estructura de la base de datos. El primer componente principal es, en esencia, una nueva variable que mide la proximidad de los individuos al centro de masas. En efecto:

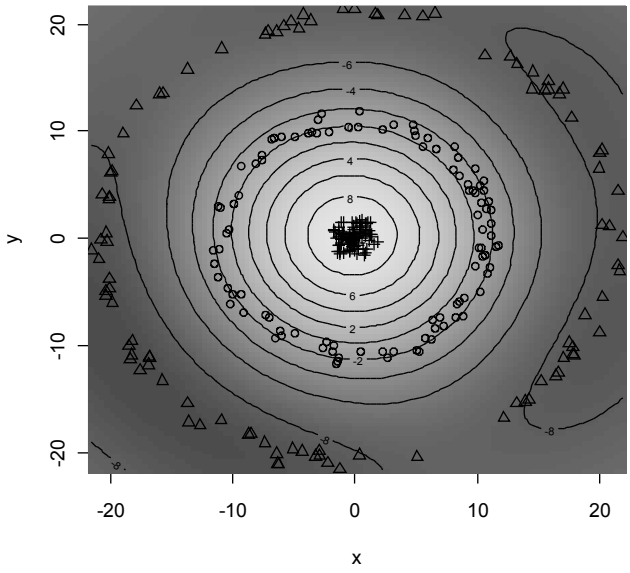
- Los elementos de la clase 1 (cruces), que en la representación del *input space* se encontraban muy cerca del origen, aparecen situados en el extremo derecho del gráfico anterior.

- Los elementos de la clase 3 (triángulos), que en la representación original se encontraban muy alejados del origen, aparecen en el espacio transformado ocupando los valores más bajos (extremo izquierdo) del primer componente principal.

- Por su parte, los elementos de la clase 2 (círculos), que ocupaban una posición intermedia en la representación original, siguen haciéndolo cuando son representados en el componente principal del *feature space*.

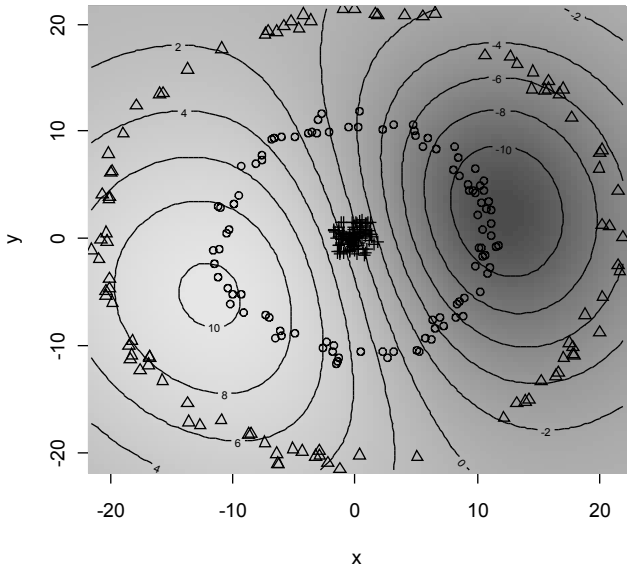
Además de esta representación de las coordenadas de los individuos en los componentes principales extraídos en el *feature space,* puede resultar de interés representar los puntos en el espacio original (*input space*) complementando dicha representación con un degradado en escala de grises y unas curvas de nivel alusivos al valor que a cada punto del *input space* le corresponde cuando su imagen por el *embedding* es proyectada sobre cada uno de los dos componentes principales.

Podemos conseguir este objetivo empleando el código de R recogido en el anexo R21 (para el primer componente principal). El resultado es el que seguidamente se observa:



Los valores del primer componente principal correspondientes a los puntos que conforman el *input space* (que vienen indicados por el degradado en escala de grises y por las curvas de nivel) se adaptan a la perfección a la estructura de la base de datos. En efecto, como se puede apreciar, todos los elementos de la clase 1 (cruces) se encuentran situados en la zona del *input space* a la que le corresponden coordenadas superiores a 8 en el primer componente principal (zona central). Por su parte, los elementos de la clase 2 (círculos) se encuentran en la zona del *input space* a la que le corresponden coordenadas situadas entre -4 y 2 en el primer componente principal (zona intermedia). Finalmente, los puntos correspondientes a la clase 3 (triángulos) se ubican en la zona del *input space* a la que le corresponden valores en el primer componente principal inferiores a -6 (zona exterior).

En el caso del segundo componente principal el código de R se recoge en el anexo R22. El resultado es el gráfico siguiente (de interpretación mucho más confusa):



Para finalizar con esta primera aplicación del *kernel PCA* vamos a mostrar, a continuación, cómo la sustitución del producto escalar ordinario por una función *kernel* adecuada es, en esencia, una alteración de la forma de medir las distancias en el *input space*.

En efecto, si las coordenadas originales de los individuos se recogen en una matriz X de l filas y n columnas (donde l es el número de individuos y n es la dimensión del *input space*) entonces la matriz $X X^T$ tiene como elemento genérico el producto escalar entre las coordenadas de los elementos *i*-ésimo e *i'*-ésimo, es decir:

$$\left(X X^T\right)_{ii'} = \langle x_i, x_{i'} \rangle$$

A partir de los elementos que componen la matriz $X X^T$ es muy sencillo calcular cuál es la distancia entre cualquier pareja de elementos. En efecto, es fácil comprobar que

$$d^2(x, z) = \|x - z\|^2 = \langle x, x \rangle - 2\langle x, z \rangle + \langle z, z \rangle$$

Sustituir el producto escalar ordinario por una función *kernel* determinada es equivalente a sustituir la matriz $X X^T$ por la matriz *kernel* K. Ahora, las "nuevas distancias[25]" entre los elementos (inducidas por la función *kernel* elegida) vendrán dadas por la siguiente expresión:

$$\|\phi(x) - \phi(z)\|^2 = k(x, x) - 2k(x, z) + k(z, z)$$

Es decir, las nuevas distancias entre los elementos se pueden calcular de forma muy simple a partir de la información recogida en la matriz *kernel* K.

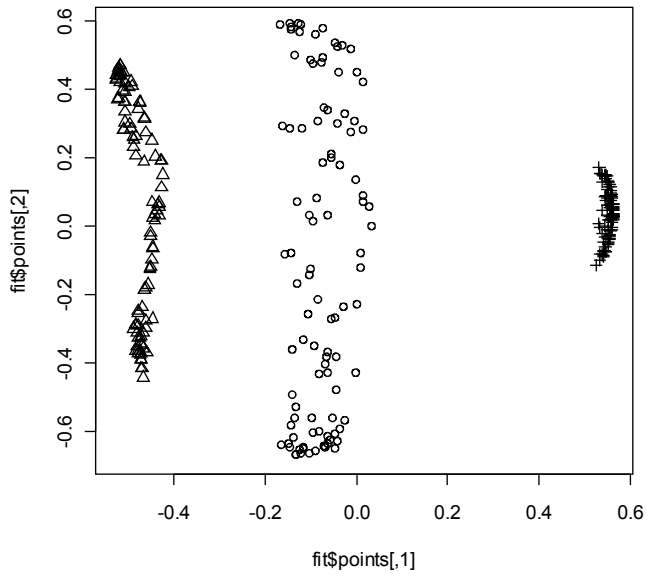
Para los datos del ejemplo se puede utilizar el código de R del anexo R23.

25 Mejor dicho, los cuadrados de las nuevas distancias.

El resultado de este código es la matriz *kernel* K (recogida en el objeto `KERNEL`) y la matriz de "nuevas distancias" entre los elementos de la muestra (recogida en el objeto `DIST`). Ambas matrices tienen dimensión 300×300, lo que impide su completa visualización. Afortunadamente existen métodos diseñados para representar gráficamente una matriz de distancias. Podemos, por ejemplo, utilizar el *escalamiento multidimensional* para tratar de incrustar los elementos en $\mathbb{R}^2$ a partir de la información acerca de las distancias existentes entre ellos. Obtenemos, de esta forma, una representación gráfica de la matriz de distancias, ya que los elementos se disponen en el plano de manera que reflejen de modo óptimo las verdaderas distancias (nuevas distancias inducidas por la función *kernel)* que existen entre ellos.

Podemos llevar a cabo un escalamiento multidimensional de los datos para las nuevas distancias mediante el conjunto de comandos de R del anexo R24.

El resultado es el siguiente gráfico, que de forma nada sorprendente, se asemeja en gran medida al que recoge las coordenadas de los elementos del ejemplo sobre los dos primeros componentes principales del *feature space*:



Esta forma de proceder sugiere un método para *kernelizar* cualquier procedimiento de análisis de datos cuyo *input* sea una matriz de distancias[26]. En efecto, bastará con sustituir la matriz de distancias habitual por la matriz de las distancias inducidas por la función *kernel.* Como se ha puesto de manifiesto con este ejemplo, el cálculo de la matriz de "nuevas distancias" a partir de la matriz *kernel* es trivial.

26 Para profundizar en este tema puede consultarse Schölkopf (2001).

Aplicación del package `kernlab` al análisis de la base de datos sobre variables de competitividad en regiones europeas

Tras el análisis de la base de datos artificial construida *ad hoc* procedemos, por último, a la aplicación de la técnica de *kernel PCA* a una base de datos real. Se trata, en concreto, de un subconjunto de la base de datos de regiones europeas que se analizó en el apartado dedicado a las *support vector machines*.

Los datos del ejemplo están recogidos en el archivo *ExampleKernelPCA.csv*. Los comandos para leer los datos desde R, tipificar las variables (dada su diferente variabilidad) y obtener sus características básicas se encuentran en el anexo R25.

Las características básicas del conjunto de datos son las siguientes:

```
> dim(datos)
[1] 188   3
> summary(datos)
   UE15         GDPpc               Agric
 UE10: 41   Min.   :-1.641e+00   Min.   :-1.006e+00
 UE15:147   1st Qu.:-6.910e-01   1st Qu.:-6.510e-01
            Median : 1.287e-01   Median :-3.454e-01
            Mean   :-1.175e-16   Mean   :-8.350e-18
            3rd Qu.: 5.628e-01   3rd Qu.: 2.811e-01
            Max.   : 4.113e+00   Max.   : 4.247e+00
```

En resumen, contamos con una base de datos en la que se recogen los valores de 188 individuos (regiones de la Unión Europea) en un total de tres variables:

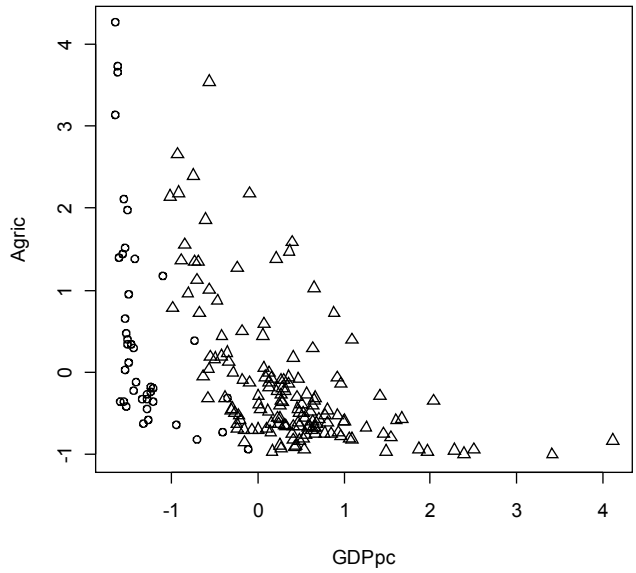
- Una etiqueta que, con carácter puramente ilustrativo[27], identifica a las regiones como miembros de la UE-15 o de la UE-10, según el criterio que se expuso en el apartado dedicado a las *support vector machines* y que, por brevedad, se omitirá en este caso.

- La variable `GDPpc`, que mide el *Producto Interior Bruto per cápita* en la región. Es una medida básica de su productividad y, por ende, de su competitividad.

- La variable `Agric`, que mide el peso del sector agrícola en la economía de la región como porcentaje del empleo total que se encuentra asignado a dicho sector de actividad.

El diagrama de dispersión de las regiones en función de las dos variables cuantitativas e identificadas en función de su adscripción a la UE-15 o a la UE-10 es el siguiente:

27 Como ya se comentó en el ejemplo anterior, la etiqueta tiene como objetivo facilitar la comparación entre las representaciones de los individuos.



En general, las regiones de la UE-15 (representadas por triángulos) se encuentran más próximas a la esquina inferior derecha del diagrama de dispersión (asociadas a un PIB per cápita relativamente alto así como a un menor peso relativo de la agricultura en el total de la economía de la región) mientras que las regiones de la UE-10 (círculos) se sitúan más próximas a la esquina superior izquierda, lo que indica valores relativamente bajos en PIB per cápita y relativamente altos en lo que al peso del sector agrícola se refiere.

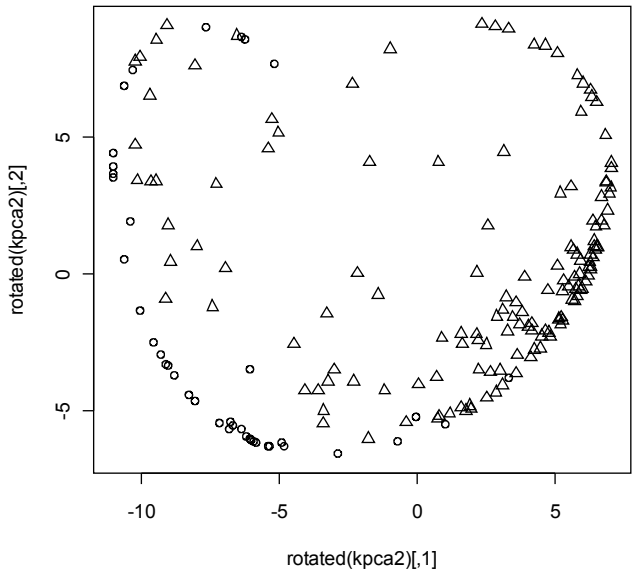
Sobre este conjunto de datos, vamos a aplicar la técnica del *kernel PCA* empleando de nuevo un *kernel* gaussiano de parámetro gamma=0,25 (establecido de nuevo por el método de *ensayo y error*, ya que el *package* `kernlab` carece de una función equivalente a la función `tune()` del *package* `e1071`).

El código empleado para este fin se presenta a continuación:

```
kpca2<-kpca(~.,as.data.frame(datos[,2:3]),kernel="rbfdot",kpar=list(sigma=0.25),features=2)
```

Los resultados (acompañados de las funciones que los generan) son los siguientes:

```
> eig(kpca2)
   Comp.1    Comp.2
0.1862861 0.1012458
> plot(rotated(kpca2),pch=as.integer(datos[,1]))
```

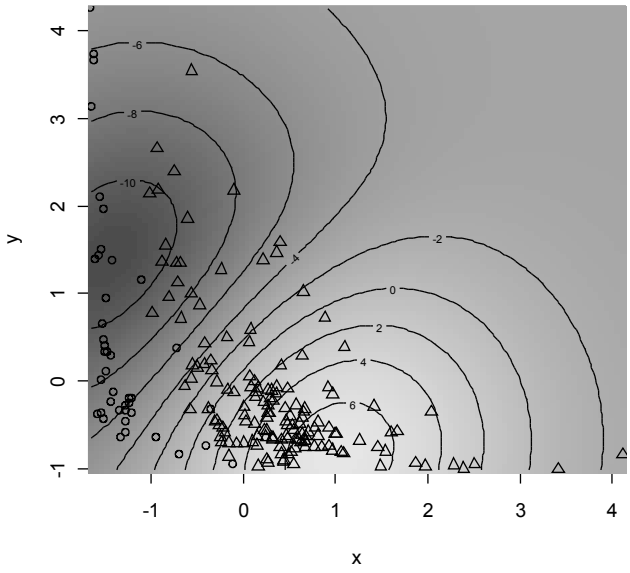


El gráfico muestra cómo esta pareja de componentes principales extraídos en el *feature space* identifica con bastante fidelidad la estructura que subyace a la base de datos (en todo caso, lo hace mucho mejor que el diagrama de dispersión para la representación de los puntos en el *input space*). En efecto:

- Las regiones de la UE-10 (círculos) quedan prácticamente confinadas en el extremo izquierdo del gráfico (es decir, con valores bajos en el primer componente principal) si bien algunas alcanzan valores intermedios en este primer componente. En este último caso, los valores en el segundo componente principal son muy reducidos.

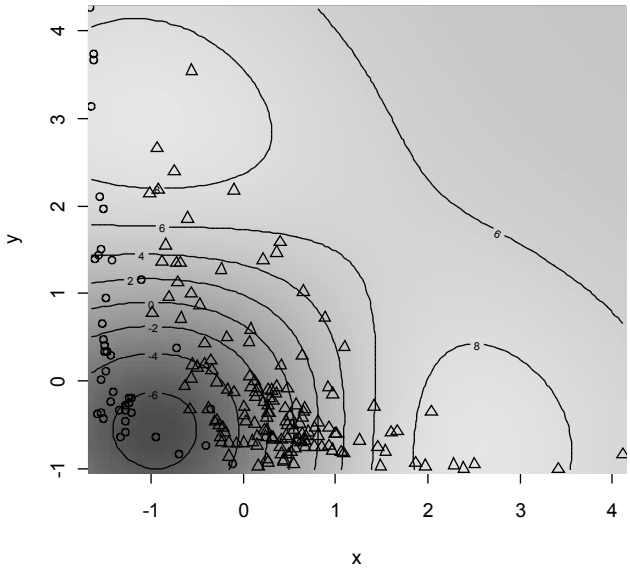
- Las regiones de la UE-15 (triángulos) ocupan la parte derecha del gráfico, es decir, tienen asignadas coordenadas muy elevadas en el primer componente principal extraído en el *feature space*. Algunas, no obstante, ocupan la parte central del gráfico.

Una vez más, puede resultar ilustrativo representar los individuos en el *input space* pero acompañándolos del valor que a cada punto de dicho espacio le corresponde cuando su imagen por el *embedding* implícitamente definido por el *kernel* gaussiano se proyecta sobre los primeros componentes principales extraídos en el *feature space*. El código necesario para obtener este gráfico (para el primer componente principal) se encuentra en el anexo R26, cuyo resultado es el siguiente:



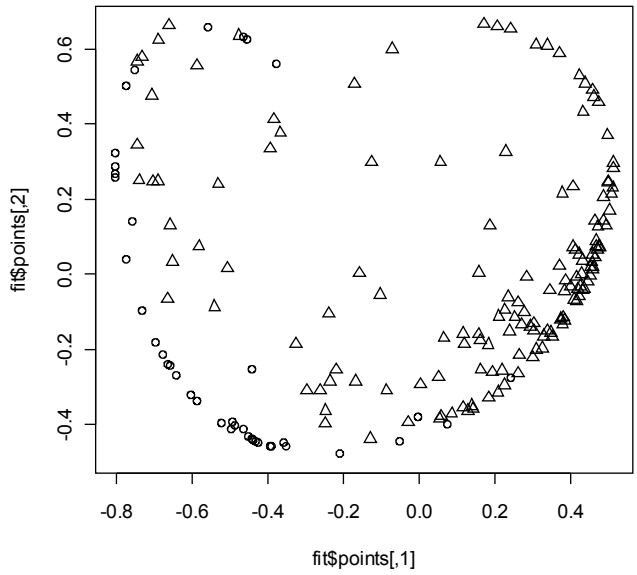
En el gráfico se aprecia cómo los puntos correspondientes a las regiones de la UE-15 (triángulos) aparecen mayoritariamente situados en aquellos lugares del *input space* a los que les corresponden valores relativamente elevados en el primer componente principal del *kernel PCA* (zona más clara situada en las proximidades de la esquina inferior derecha), mientras que las regiones de la UE-10 (círculos) están primordialmente situadas en la zona del *input space* a la que le corresponden valores bajos en el primer componente principal (zona más oscura, situada en las proximidades de la esquina superior izquierda). Así pues, este primer componente principal del *kernel PCA* recoge, al menos de forma rudimentaria, la estructura subyacente a la base de datos.

En el caso del segundo componente principal, el código de R se encuentra en el anexo R27 y el gráfico resultante de su ejecución (de interpretación más compleja) es el siguiente:



Finalmente, al igual que hicimos con el ejemplo de la base de datos ficticia, podemos tratar de reconstruir el diagrama de dispersión de las regiones en el *feature space* a través de un *escalamiento multidimensional* a partir de la matriz de "nuevas distancias" calculadas tomando como *input* la matriz *kernel* resultante de aplicar el *kernel* elegido a todas las parejas de elementos. Para ello debemos ejecutar el conjunto de comandos de R del anexo R28.

El resultado es el gráfico siguiente, muy similar al que presenta las coordenadas de los elementos sobre los dos primeros componentes principales del *feature space*:



De esta forma damos por finalizada la aplicación de la teoría del *kernel PCA*. Para ilustrar esta técnica se ha recurrido al *package* `kernlab` de R y se han analizado un par de bases de datos: la primera, una base creada *ad hoc*, que ha permitido poner de manifiesto la capacidad del *kernel PCA* para sacar a la luz estructuras en los datos que resultan invisibles para el análisis de componentes principales tradicional; la segunda, una base de datos sobre variables de competitividad en regiones europeas, que ha supuesto la aplicación del *kernel PCA* a un caso real.

En este tema se han propuesto representaciones gráficas adicionales a las que proporciona el *package* `kernlab` de R. En concreto, se ha representado a los individuos en el *input space* acompañados de una escala de grises alusiva al valor que correspondería a cada punto del *input space* cuando su imagen por el *embedding* se proyecta sobre los componentes principales extraídos en el *feature space*. Asimismo, se ha facilitado el código de R para obtener tales representaciones gráficas.

Finalmente, se ha esbozado una vía para *kernelizar* cualquier algoritmo de análisis de datos cuyo *input* sea una matriz de distancias (como es el caso del escalamiento multidimensional). La idea básica consiste en que la sustitución del producto escalar estándar por una función *kernel* válida es, en esencia, una modificación de la forma de medir las distancias en el *input space*. Además, el cálculo de estas "nuevas distancias" entre los elementos es inmediata a partir de la información contenida en la matriz *kernel.*

Como colofón al trabajo, se presentan unos comentarios finales.

Comentario final

Alcanzado el final del documento, que constituye el *Trabajo de Fin de Máster* del *Máster en Matemáticas Avanzadas* de la Facultad de Ciencias de la UNED, procede pasar revista a los objetivos cumplidos así como a aquello que ha quedado pendiente.

Con este trabajo se ha pretendido:

- Exponer el cambio de paradigma que se está observando dentro del mundo científico, que está asistiendo a un desplazamiento del centro de gravedad desde los modelos hacia los datos. En este sentido, se ha pretendido destacar el papel que, en este nuevo paradigma, jugará el desarrollo de algoritmos de detección de patrones.

- Describir los objetivos básicos de los algoritmos de detección de patrones así como de la disciplina del aprendizaje estadístico. En concreto, se ha pretendido mostrar las ventajas e inconvenientes de las técnicas lineales y no lineales de detección de patrones, tanto en el ámbito supervisado como en el no supervisado.

- Analizar la adecuación de la solución *kernel* para tratar de reconciliar el dilema entre lo lineal y lo no lineal. En este documento se ha propuesto la solución *kernel* como una inteligente vía para integrar lo mejor de estos dos mundos.

- Elegida la solución *kernel*, se ha pretendido realizar una recopilación de la bibliografía más relevante en este ámbito y, muy en particular, en el campo de las *support vector machines*, que constituyen la solución *kernel* más asentada y estudiada.

- Asimismo, se ha perseguido mostrar con detalle cómo es posible (y sencillo) *kernelizar* algunos algoritmos lineales de detección de patrones: se han adaptado, para su utilización con funciones *kernel,* un clasificador *hard* y un clasificador *soft* (dentro del aprendizaje de tipo supervisado) así como el análisis de componentes principales (dentro del aprendizaje no supervisado). Se han dado, además, algunas indicaciones sobre cómo proceder en el caso de algoritmos que tengan como *input* una matriz de distancias (caso del análisis clúster o el escalamiento multidimensional).

- Para cada uno de estos algoritmos *kernelizados* se han presentado ejemplos de diversos tamaños: ejemplos de ilustración (dentro de las exposiciones teóricas), ejemplos construidos *ad hoc* (ficticios) y ejemplos de investigaciones reales (caso de las regiones de la UE y su competitividad).

- Estos ejemplos se han resuelto, en todos los casos, con el entorno de programación R, por su potencia y versatilidad. En concreto, se han empleado los *packages* kernlab, e1071 y FactoMineR.

Lamentablemente, algunos de los objetivos que inicialmente se contemplaron como interesantes no han tenido cabida finalmente en el documento:

- No se ha considerado el aprendizaje semi-supervisado.

- Sólo se han analizado las *support vector machines* en el caso de clasificación en dos grupos. Se ha dejado de lado el caso de clasificación en más de dos grupos y el caso de la regresión.

- Aunque se ha indicado cómo podría aplicarse el método basado en funciones *kernel* a aquellos algoritmos lineales cuyo *input* es una matriz de distancias, esta exposición no ha sido detallada.

- No se han estudiado las versiones *sparse* de los métodos *kernelizados* presentados.

- Se ha optado por el entorno de programación R sin exponer las ventajas e inconvenientes de éste frente a otras alternativas (Weka, Matlab,...)

Antes de finalizar, un último comentario. Según Zhu (2008), "[...] los algoritmos basados en funciones *kernel* son como cámaras fotográficas profesionales. Son capaces de producir fotografías de gran calidad incluso en circunstancias muy difíciles, pero es necesario ponerlas en manos de fotógrafos profesionales. Si se las das a un aficionado o a un novato no puedes esperar fotografías aceptables. El fotógrafo debe saber cómo seleccionar la lente adecuada, establecer la velocidad de obturación correcta y fijar la apertura correspondiente a cada condición posible. Si alguno de estos parámetros no se fija apropiadamente, el resultado podría ser un desastre". Siguiendo con este símil, el presente trabajo es el esfuerzo de un aficionado que quiere comprender cómo funciona la cámara que acaban de poner en sus manos, y que tiene la osadía suficiente como para mostrar sus primeras "obras de arte", tanto las que ha realizado dentro de su estudio, como las que ha ido haciendo por la calle. Esperemos que el resultado no sea tan desastroso como el que anticipa Zhu.

Bibliografía

Aizerman, M.A.; Braverman, E.M.; Rozonoer, L.I.; (1964); "Theoretical foundations of the potential function method in pattern recognition learning"; *Automation and Remote Control*; Vol. 25; pp, 821-837

Anderson, C.; (2008); "The End of Theory: The Data Deluge Makes the Scientific Method Obsolete"; *Wired Magazine* [http://www.wired.com/science/discoveries/magazine/16-07/pb_theory]

Aronszajn, N.; (1950); "Theory of Reproducing Kernels"; *Transactions of the American Mathematical Society*; Vol. 68; Issue 3; May, 1950 [http://web.mit.edu/anandk/OldFiles/MacData/afs.anandk/MacData/afs.course/9/9.520/www/Papers/aron.pdf]

Baudat, G.; Anouar, F.; (2000); "Generalized Discriminant Analysis Using a Kernel Approach"; *Neural Computation*; Vol. 12; no. 10; pp. 2385-2404 [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.1662&rep=rep1&type=pdf]

Baker, S.; (2008); The Numerati; Houghton Mifflin Company; New York

Bazaraa, M.S.; Sherali, H.D.; Shetty, C.M.; (2006); Nonlinear Programming: Theory and Algorithms; 3rd edition; John Wiley and Sons; Hoboken; New Jersey

Ben-Hur, A.; Ong, C.S.; Sonnenburg, S.; Schölkopf, B.; Rätsch, G.; (2008); "Support Vector Machines and Kernels for Computational Biology"; *PloS Computational Biology* (www.ploscompbiol.org); Vol. 4; Issue 10; October 2008 [http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2547983/pdf/pcbi.1000173.pdf]

Bennett, K.P.; Campbell, C.; (2000); "Support Vector Machines: Hype or Hallelujah?"; *SIGKDD Explorations;* Vol. 2; Issue 2; p.1 [http://www.idi.ntnu.no/emner/it3704/lectures/papers/Bennett_2000_Support.pdf]

Bennett, K.P.; Mangasarian, O.L.; (1992); "Robust linear programming discrimination of two linearly inseparable sets"; *Optimization Methods and Software;* Vol. 1; pp. 23–34 [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.23.3307&rep=rep1&type=pdf]

Boser, B.E.; Guyon, I.M.; Vapnik, V.N.; (1992); "A training algorithm for optimal margin classifiers"; *Annual Workshop on Computational Learning Theory*; Pittsburgh; Pennsylvania; pp. 144-152 [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.3818&rep=rep1&type=pdf]

Boyd, S.; Vandenberghe, L.; (2004); Convex Optimization; Cambridge University Press; New York

Breiman, L.; (2001); "Statistical Modeling: the Two Cultures"; *Statistical Science;* Vol. 16; nº 3; pp. 199-231 [http://www.ics.uci.edu/~dgillen/Stat235/Handouts/breiman2001.pdf]

Burges, C.J.C.; (1998); "A Tutorial on Support Vector Machines for Pattern Recognition"; *Data Mining and Knowledge Discovery;* pp. 121-167; Kluwer Academic Publishers; Boston [http://www.springerlink.com/content/q87856173126771q/fulltext.pdf]

Burges, C.J.C.; (2005); “Geometric Methods for Feature Extraction and Dimensional Reduction: A Guided Tour”: en *The Data Mining and Knowledge Discovery Handbook*; Maimon, O.Z.; Rokach, L. (eds.); Springer [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.71.2181&rep=rep1&type=pdf]

Chang, C.C.; Lin, C.J. (2010a); LIBSVM: a Library for Support Vector Machines [http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf]

Chang, C.C.; Lin, C.J. (2010b); A Practical Guide to Support Vector Classification; [http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf]

Cortes, C.; Vapnik, V.; (1995); “Support-Vector Networks”; *Machine Learning*; Vol. 20; pp. 273-297 [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.15.9362&rep=rep1&type=pdf]

Cover, T.M.; (1965); “Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition”; *IEEE Transactions on Electronic Computers*; Vol. EC-14; no. 3; pp. 326-334 [http://www.stanford.edu/~cover/papers/paper2.pdf]

Cristianini, N.; Shawe-Taylor, J.; (2000); An Introduction to Support Vector Machines and other Kernel-based Learning Methods; Cambridge University Press

Dimitriadou, E.; Hornik, K.; Leisch, F.; Meyer, D.; Weingessel, A.; (2010); e1071: Misc Functions of the Department of Statistics (e1071), TU Wien. R package version 1.5-24. http://CRAN.R-project.org/package=e1071

Dreyfus, G.; Martinez, J.M.; Samuelides, M.: Gordon, M.B.; Badran, F.; Thiria, S.; (2008); Apprentissage statistique; Éditions Eyrolles, Paris

Duda, R.O.; Hart, P.E.; (1973); Pattern Classification and Scene Analysis; John Wiley & Sons; New York

Eldén, L.; (2007); Matrix Methods in Data Mining and Pattern Recognition; Society for Industrial and Applied Mathematics; Philadelphia

Fisher, R.A.; (1936); “The use of multiple measurements in taxonomic problems”; *Annals of Eugenics*; Vol. 7; pp. 179-188 [digital.library.adelaide.edu.au/coll/special/fisher/138.pdf]

Girolami, M.; (2001); “Mercer Kernel Based Clustering in Feature Space”; *IEEE Transactions on Neural Networks*; Vol. 13; pp. 780-784 [http://www.kernel-machines.org/papers/upload_23061_tnnl0049_df.zip]

Hamel, L.; (2009); Knowledge Discovery with Support Vector Machines; John Wiley and Sons; Hoboken; New Jersey

Hastie, T.; Tibshirani, R.; Friedman, J.; (2009); The Elements of Statistical Learning: Data Mining, Inference and Prediction; 2nd edition; Springer [http://www-stat.stanford.edu/~hastie/local.ftp/Springer/ESLII_print3.pdf]

Hein, M.; (2005); Geometrical Aspects of Statistical Learning Theory; PhD Thesis [http://tuprints.ulb.tu-darmstadt.de/673/1/Hein-

Geometrical_Aspects_of_Statistical_Learning_Theory.pdf]

Hey, T.; Tansley, S.; Tolle, K.; (2009); The Fourth Paradigm: Data-Intensive Scientific Discovery; Microsoft Research; Redmond; Wahington [http://research.microsoft.com/en-us/collaboration/fourthparadigm/4th_paradigm_book_complete_lr.pdf]

Hindi, H.; (2004); "A Tutorial on Convex Optimization"; *American Control Conference*, 2004, 30 June – 2 July 2004 [http://www2.parc.com/spl/members/hhindi/reports/CvxOptTutPaper.pdf]

Hindi, H.; (2006); "A Tutorial on Convex Optimization II: Duality and Interior Point Methods"; *American Control Conference*, 2006; 14-16 June 2006; Minneapolis, MN [http://cosmal.ucsd.edu/~gert/ECE273/hindiTutorial2.pdf]

Hofmann, T.; Schölkopf, B.; Smola, A.J.; (2008); "Kernel methods in machine learning"; *The Annals of Statistics*; Vol. 36; Number 3 (2008); pp. 1171-1220 [http://arxiv.org/PS_cache/math/pdf/0701/0701907v3.pdf]

Huang, T.M.; Kecman, V.; Kopriva, I.; (2006); Kernel Based Algorithms for Mining Huge Data sets: Supervised, Semi-supervised and Unsupervised Learning; Springer-Verlag; Berlin

Husson, F.; Josse, J.; Le, S.; Mazet, J.; (2010); FactoMineR: Multivariate Exploratory Data Analysis and Data Mining with R. R package version 1.14. http://CRAN.R-project.org/package=FactoMineR

Izenman, A.J.; (2008); Modern Multivariate Statistical Techniques: Regression, Classification and Manifold Learning; Springer

Izenman, A.J.; Shen, Y.; (2008); "Outlier Detection Using the Smallest Kernel Principal Components"; *JSM-2008* [http://astro.ocis.temple.edu/~alan/OutlierDetectionpaper.PDF]

Jollife, I.T.; (2002); Principal Components Analysis; 2nd edition; Springer-Verlag; New York

Karatzoglou, A.; Smola,A.; Hornik, K.; Zeileis, A.; (2004); "kernlab - An S4 Package for Kernel Methods in R"; *Journal of Statistical Software*; 11(9), 1-20. URL http://www.jstatsoft.org/v11/i09/

Lanckriet, G.R.G.; Cristianini, N.; Bartlett, P.; El Ghaoui, L.; Jordan, M.I.; (2004); "Learning the Kernel Matrix with Semidefinite Programming"; *Journal of Machine Learning Research* 5 (2004) 27-72 [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.6732&rep=rep1&type=pdf]

Mangasarian, O.L.; (1965); Linear and Nonlinear Separation of Patterns by Linear Programming; *Operations Research*; Vol. 13; Issue 3; May-June, 1965; pp. 444-452 [http://pages.cs.wisc.edu/~olvi/oldpapers/olmor.pdf]

Mavroforakis, M.E.; Theodoridis, S.; (2006); "A Geometric Approach to Support Vector Machine (SVM) Classification"; *IEEE Transactions on Neural Networks*; Vol. 17; Issue 3; pp. 671-682 [http://cgi.di.uoa.gr/~idsp/Signal%20and%20Image%20Processing%20Files/papers/mavroforakis/SVM-Geom-IEEE-01-printed.pdf]

Mercer, J.; (1909); Functions of Positive and Negative Type, and their Connection with the Theory of Integral Equations; *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*; vol. 209; pp. 415-446

Navarro, M.; Gibaja, J. J.; Bilbao-Osorio, B.; Aguado, R.; (2009); "Patterns of innovation in EU-25 regions: a typology and policy recommendations"; *Environment and Planning C: Government and Policy*; Vol. 27; Issue 5; pp. 815–840 [http://www.orkestra.deusto.es/eunip2008/wp-content/uploads/2008/10/navarrogibaja-bilbaoaguado.pdf]

Okutmustur, B.; (2005); Reproducing Kernel Hilbert Spaces; Master Thesis; Institute of Engineering and Science; Bilkent University [http://www.thesis.bilkent.edu.tr/0002953.pdf]

Paulos, J.A.; (1990); Innumeracy: Mathematical Illiteracy and Its Consequences; Econo-Clad Books.

Poggio, T.; Girosi, F.; (1990); "Networks for approximation and learning"; *Proceedings of the IEEE*; Vol. 78; Issue 9; pp. 1481–1497 [http://courses.cs.tamu.edu/rgutier/cpsc636_s10/poggio1990rbf2.pdf]

R Development Core Team; (2010); R: A language and environment for statistical computing. R Foundation for Statistical Computing; Vienna, Austria; ISBN 3-900051-07-0; URL http://www.R-project.org.

Resendiz Trejo, J.A.; (2006); Las máquinas de vectores de soporte para identificación en línea; Tesis de Máster; Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional [http://www.ctrl.cinvestav.mx/~yuw/pdf/MaTesJAR.pdf]

Sánchez, L.G.; Osorio, G.A.; Suárez, J.F.; (2008); "Introducción a kernel ACP y otros métodos espectrales aplicados al aprendizaje no supervisado"; *Revista Colombiana de Estadística*; Vol. 31; no. 1; pp. 19-40 [ftp://ftp.math.ethz.ch/hg/EMIS/journals/RCE/body/v31n1a02SanchezOsorioSuarez.pdf]

Schölkopf, B.; (2001); "The Kernel Trick for Distances"; *Advances in Neural Information Processing Systems 13*; pp. 301-307; Leen, T.K.; Dietterich, T.G.; Tres, V.; MIT Press; Cambridge; MA [http://www.kyb.mpg.de/publications/attachments/scholkopf00kernel_3781%5B0%5D.pdf]

Schölkopf, B.; Burges, C.J.C.; Smola, A.J.; (1998); Advances in Kernel Methods: Support Vector Learning; The MIT Press; Cambridge; Massachusetts

Schölkopf, B.; Mika, S.; Burges, C.J.C.; Knirsch, P.; Müller, K.R.; Rätsch, G.; Smola, A.J.; (1999); "Input Space Versus Feature Space in Kernel-Based Method"; *IEEE Transactions on Neural Networks*; Vol. 10; no. 5; September 1999 [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.4451&rep=rep1&type=pdf]

Schölkopf, B.; Smola, A.J.; (2003); "A Short Introduction to Learning with Kernels" en *Advanced Lectures on Machine Learning*; Mendelson, S.; Smola, A.J. (eds.); LNAI 2600; pp. 41-64; Springer-Verlag; Berlin; 2003 [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.4786&rep=rep1&type=pdf]

Schölkopf, B.; Smola, A.J.; (2002); Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond; The MIT Press; Cambridge; Massachusetts

Schölkopf, B.; Smola, A.J.; Müller, K.R.; (1996); Nonlinear Component Analysis as a Kernel Eigenvalue Problem; Max Planck Institut für biologische Kybernetik; Technical Report nº 44;

December 1996

Schölkopf, B.; Smola, A.J.; Müller, K.R.; (1998); "Nonlinear Component Analysis as a Kernel Eigenvalue Problem"; *Neural Computation*; Vol. 10; No. 5; pp. 1299-1319 [http://www.kyb.tuebingen.mpg.de/people/personal/bs/papers/kpca_nc.ps.gz]

Schrodt, P.A.; (2010); "Seven Deadly Sins of Contemporary Quantitative Political Analysis"; *APSA 2010 Annual Meeting* [http://ssrn.com/abstract=1661045]

Smith, F.W.; (1968); "Pattern Classifier Design by Linear Programming"; *IEEE Transactions on Computers*; vol. c-17; no. 4; April 1968 [http://www.computer.org/portal/web/csdl/doi/10.1109/TC.1968.229395]

Shawe-Taylor, J.; Bartlett, P.L.; Williamson, R.C.; Anthony, M.; (1998); "Structural Risk Minimization over Data-Dependent Hierarchies; *IEEE Transactions on Information Theory*; Vol. 44; Issue 5; pp. 1926-1940 [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.2159&rep=rep1&type=pdf]

Shawe-Taylor, J.; Cristianini, N.; (2000); "Margin distribution and soft margin"; En *Advances in Large Margin Classifiers;* Smola, A.J. *et al., eds. ;* The MIT Press; Cambridge; Massachusetts; pp. 349–358

Shawe-Taylor, J.; Cristianini, N.; (2004); Kernel Methods for Pattern Analysis; Cambridge University Press; New York

Smola, A.J.; Bartlett, P.L.; Schölkopf, B.; Schuurmans, D.; (2000); Advances in Large-Margin Classifiers; The MIT Press; Cambridge; Massachusetts

Suykens, J.A.K.; Van Gestel, T.; Vandwalle, J.; De Moor, B.; (2003); "A Support Vector Machine Formulation to PCA Analysis and Its Kernel Version"; *IEEE Transactions on Neural Networks*, Vol. 14; no. 2; March 2003 [ftp://ftp.esat.kuleuven.ac.be/sista/ida/reports/02-68.pdf]

Theodoridis, S.; Koutroumbas, K.; (2003); Pattern Recognition; 2nd edition; Elsevier

The Economist; (2010); Data, data everywhere: A Special Report on Managing Information; 27 February 2010 [http://pages.cs.wisc.edu/~sblanas/EconomistOnData.pdf]

Van Gestel, T.; Suykens, J.A.K.; De Brabanter, J.; De Moor, B.; Vandewalle, J.; (2001); "Kernel Canonical Correlation Analysis and Least Squares Support Vector Machines"; *Artificial Neural Networks ICANN 2001* [ftp://ftp.esat.kuleuven.ac.be/pub/sista/suykens/reports/lssvm_01_24.ps.gz]

Vanderbei, R.J.; (1988); LOQO: "An Interior Point Code for Quadratic Programming"; *Optimization Methods and Software*; Vol. 11; Issue 1-4; pp. 451-484 [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.2191&rep=rep1&type=pdf]

Vapnik, V.N.; (1979); Estimation of Dependences Based on Empirical Data [en ruso]; Nauka; Moscú

Vapnik, V.N.; (1982); Estimation of Dependences Based on Empirical Data; Springer

Vapnik, V.N. (1998); Statistical Learning Theory; John Wiley and Sons

Vapnik, V.N.; (1995); The Nature of Statistical Learning Theory; Springer; New York

Vapnik, V.N.; (1999); The Nature of Statistical Learning Theory; 2nd edition; Springer; New York

Vapnik, V.N.; Chervonenkis, A.; (1964); "A note on one class of perceptrons"; *Automation and Remote Control*, Vol. 25

Vapnik, V.N.; Chervonenkis, A.; (1974); Teoriya raspoznavaniya obrazov: Statisticheskie problemy obucheniya. (Ruso) [Theory of pattern recognition: Statistical problems of learning]. Nauka; Moscú

Vapnik, V.N.; Lerner, A.; (1963); "Pattern Recognition using Generalized Portrait Method"; *Automation and Remote Control*; Vol. 24; pp. 774-780

Zhu, M. (2008); "Kernels and Ensembles"; *The American Statistician*. May 2008; Vol. 62; No. 2; pp. 97-109 [http://arxiv.org/PS_cache/arxiv/pdf/0712/0712.1027v1.pdf]

Ziliak, S.T.; McCloskey, D.N.; (2008); The Cult of Statistical Significance: How the Standard Error Costs Us Jobs, Justice, and Lives; The University of Michigan Press; Ann Arbor.

Recursos *online*

http://www.kernel-machines.org/

http://www.support-vector-machines.org/SVM_soft.html

http://www.csie.ntu.edu.tw/~cjlin/libsvm/

http://cran.r-project.org/web/packages/kernlab/index.html (paquete kernlab de R)

http://cran.r-project.org/web/packages/e1071/index.html (paquete e1071 de R)

http://cran.r-project.org/web/packages/FactoMineR/index.html (paquete FactoMineR de R)

http://www.heikohoffmann.de/htmlthesis/node36.html

http://www.kernel-methods.net/

http://videolectures.net/Top/Computer_Science/Machine_Learning/Kernel_Methods/

http://sml.nicta.com.au/

http://www.stat.berkeley.edu/~statlearning/

http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/TWINING1/index.html

http://www.uic.edu/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/2961/2573

http://www.svms.org/

Anexo: elementos matemáticos esenciales en la definición de las funciones *kernel*

M01: Espacio con producto interno. Espacio de Hilbert

Un espacio vectorial X sobre $\mathbb{R}$ es un *espacio con producto interno* (*inner product space*) si existe un producto interno (*inner product*) definido en X es decir, si existe una función

$$<,>: X \times X \to \mathbb{R}$$

que cumple:

1. $\langle x+y, z\rangle = \langle x, z\rangle + \langle y, z\rangle$
2. $\langle \alpha x, z\rangle = \alpha \langle x, z\rangle$
3. $\langle x, z\rangle = \langle z, x\rangle$
4. $\langle x, x\rangle \geq 0 \quad y \quad \langle x, x\rangle = 0 \quad sii \quad x = 0$

con $x, y, z \in X$ y $\alpha \in \mathbb{R}$

Todo espacio con producto interno es un *espacio métrico*, en el que la distancia se define del siguiente modo:

$$d(x, z) = \|x - z\| = \langle x - z, x - z\rangle^{1/2}$$

Además, todo espacio métrico dispone de una *topología derivada* cuya sub-base viene dada por el conjunto de las bolas abiertas del espacio. Por tanto, todo espacio métrico es un *espacio topológico*.

Se dice que un espacio topológico es *separable* si contiene un conjunto contable denso. Por ejemplo, el espacio vectorial $\mathbb{R}^n$ dotado de la topología euclídea es un espacio topológico separable ya que contiene a $\mathbb{Q}^n$, que es contable y denso en $\mathbb{R}^n$, ya que en todo abierto de $\mathbb{R}^n$ existen puntos de $\mathbb{Q}^n$.

Se dice que un espacio métrico es *completo* si todas las sucesiones de Cauchy de elementos del espacio son convergentes, es decir, tienen un límite que pertenece a dicho espacio métrico.

Espacio de Hilbert: un espacio con producto interno, separable y completo es un *espacio de Hilbert*.

Son ejemplos de espacios de Hilbert:

- El espacio euclídeo $\mathbb{R}^n$ con el producto escalar estándar $\langle u, v\rangle = u^T v$, cuya dimensión es $n \in \mathbb{N}$.
- El espacio L^2 de todas las funciones reales de variable real tales que la integral de f^2 sobre toda la recta real es finita. En este caso, el producto interno viene dado por:

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(x)\, g(x)\, dx$$

La dimensión del espacio L^2 es infinita.

Con la propiedad de separabilidad (que no siempre se exige para los espacios de Hilbert) podemos afirmar que todo espacio de Hilbert de dimensión finita es isomorfo a $\mathbb{R}^n$ mientras que todo espacio de Hilbert de dimensión infinita es isomorfo a L^2 .

M02: Matriz de Gram

Dado un conjunto de vectores $S = \{x_1, \ldots, x_l\}$ de un espacio con producto interno X , se llama *matriz de Gram* (*Gram matrix*) a la matriz de dimensión $l \times l$ cuyos elementos son los productos internos de todas las parejas que se pueden formar a partir del conjunto de vectores S . Si denotamos por G_{ij} al elemento genérico de la matriz de Gram (es decir, el que ocupa la fila i -ésima y la columna j -ésima) tendremos que:

$$G_{ij} = \langle x_i, x_j \rangle$$

En el caso particular en el que estemos empleando una función *kernel* $k : X \times X \to \mathbb{R}$ para definir el producto interno entre las imágenes de los elementos de S , la matriz de Gram recibirá el nombre de *matriz kernel* (*kernel matrix*), a la que habitualmente denotaremos por K , y sus elementos serán:

$$G_{ij} = K_{ij} = k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

La matriz de Gram (y también la matriz *kernel* por ser un caso particular de la anterior) son simétricas (lo que se deduce directamente del carácter simétrico que se exige al producto interno definido en el espacio de Hilbert) y juegan un papel central en la determinación de los patrones lineales en el *feature space* ya que proporcionan al algoritmo *kernelizado* toda la información que éste precisa para alcanzar su objetivo.

La matriz de Gram es invariante a las rotaciones, lo que significa que si efectuamos una rotación en el espacio X (en el que se encuentran los vectores del conjunto S) y calculamos la matriz de Gram correspondiente a S obtendremos la misma matriz de Gram que antes de efectuar la rotación. En efecto, la información recogida en la matriz de Gram se refiere, en exclusiva, a las normas de los vectores de S (que se encuentran en la diagonal principal de la matriz) y a los ángulos entre los vectores (que se localizan en los elementos situados fuera de la diagonal principal). Tanto las normas como los ángulos entre los vectores son invariantes a las rotaciones. La consecuencia inmediata de este hecho es que la matriz de Gram no consigue trasladar al algoritmo de detección de patrones lineales la información relativa a las posiciones absolutas de los vectores sino tan solo la información relativa a su distancia al origen y a las posiciones relativas entre los vectores (ángulos o distancias entre ellos). No obstante, esta información es suficiente para que el algoritmo pueda encontrar el patrón.

Además, las matrices de Gram (y, en consecuencia, las matrices *kernel*) son semidefinidas positivas. En efecto, consideremos un conjunto $S = \{x_1, \ldots, x_l\}$ de elementos de un espacio con producto interno X y sea G la matriz de Gram calculada a partir de S , cuya dimensión será $l \times l$.

Consideremos un vector cualquiera $u \in \mathbb{R}^l$. Tendremos que:

$$u^T G u = \sum_{i,j=1}^{l} u_i u_j G_{ij} = \sum_{i,j=1}^{l} u_i u_j \langle x_i, x_j \rangle = \sum_{i,j=1}^{l} \langle u_i x_i, u_j x_j \rangle = \langle \sum_{i=1}^{l} u_i x_i, \sum_{j=1}^{l} u_j x_j \rangle = \left\| \sum_{i=1}^{l} u_i x_i \right\|^2 \geq 0$$

Existe una propiedad adicional importante para las matrices de Gram: puede demostrarse que dada cualquier matriz simétrica semidefinida positiva A existe un conjunto de vectores para los que A es su matriz de Gram.

En efecto, dado que la matriz A es simétrica y semidefinida positiva, es ortogonalmente diagonalizable, y todos sus valores propios son no negativos, es decir, existen sendas matrices U y Λ tales que:

$$A = U \Lambda U^T$$

siendo U una matriz ortogonal cuyas columnas son vectores propios y unitarios de A y Λ la matriz diagonal cuyos elementos son los valores propios de A asociados a los respectivos vectores propios recogidos en la matriz U. Ahora, como los elementos de la diagonal de la matriz Λ son no negativos, podemos construir una nueva matriz diagonal, que denotaremos por $\Lambda^{1/2}$, y cuyos componentes serán las raíces cuadradas de los componentes de la matriz Λ, es decir:

$$(\Lambda^{1/2})_{ii} = (\Lambda_{ii})^{1/2}$$

De aquí resulta que:

$$\Lambda^{1/2} \Lambda^{1/2} = \Lambda$$

y, por tanto, podremos escribir:

$$A = U \Lambda U^T = U \Lambda^{1/2} \Lambda^{1/2} U^T = (\Lambda^{1/2} U^T)^T (\Lambda^{1/2} U^T)$$

(donde la tercera igualdad se deduce de la simetría de la matriz $\Lambda^{1/2}$). A partir de esta expresión, se concluye que A es la matriz de Gram para el conjunto de columnas de la matriz:

$$\Lambda^{1/2} U^T$$

consideradas como vectores.

Esta afirmación es una versión en dimensión finita del conocido como *teorema de Mercer* que, como se verá, juega un papel protagonista en la formalización teórica de los métodos de detección de patrones basados en funciones *kernel*.

Es importante destacar el papel central de la matriz *kernel* en los procedimientos de detección de patrones basados en estas funciones. La matriz *kernel* actúa como un auténtico *cuello de botella* ya que es el único medio de que se dispone para comunicar al algoritmo de detección de patrones en el *feature space*:

- la información referida a los valores de las observaciones en el *input space*, es decir, la información referida al conjunto $S=\{x_1,\ldots,x_l\}$.

- la información relativa al conocimiento experto que debe tenerse en cuenta para limitar el espacio de patrones. Recordemos que esta tarea estaba reservada a la función de *embedding* pero, dada la imposibilidad práctica de utilizarla (por la reducida eficiencia computacional que su empleo acarrearía) la tarea ha sido finalmente asignada a la función *kernel* y, por ende, a la matriz *kernel*.

Anexo: comandos de R para la resolución de los ejemplos

R01: Resolución de la formulación dual del ejemplo de ilustración de una support vector machine

```
library(kernlab)
### LECTURA DE LOS DATOS
datos<-read.csv(file="DatosSVM.csv",header=TRUE,sep=";",dec=",")
m<-dim(datos)[1]
t<-dim(datos)[2]
x<-as.matrix(datos[,-t])
y<-as.matrix(datos[,t])

### PROBLEMA DUAL
##  CALCULO DE LOS ARGUMENTOS DE ipop PARA EL PROBLEMA DUAL
C <- 5 #PARAMETRO DE COSTE
H<-(x%*%t(x))*(y%*%t(y)) # %*% ES EL PRODUCTO MATRICIAL HABITUAL; * ES EL PRODUCTO DE HADAMARD
c <- matrix(rep(-1,m))
A <- t(y)
b <- 0
l <- matrix(rep(0,m))
u <- matrix(rep(C,m))
r <- 0

##  OPTIMIZADOR CUADRÁTICO PARA EL PROBLEMA DUAL
sv2 <- ipop(c,H,A,b,l,u,r)
sv2
```

R02: Determinación de la superficie óptima de decisión en la formulación dual del ejemplo de ilustración de una support vector machine

```
##  CALCULO DE LA SUPERFICIE ÓPTIMA DE DECISIÓN
#   CÁLCULO DEL VECTOR w
w<-t(primal(sv2)*y)%*%x
w

#   CÁLCULO DE b
(primal(sv2)<0.9999*C) & (primal(sv2)>0.0001*C) # ¿CUÁLES SON SUPPORT VECTORS?
xsv<-x[(primal(sv2)<0.9999*C) & (primal(sv2)>0.0001*C),][1,] #COORDENADAS DEL 1ER S.V.
dim(xsv)<-c(t-1,1)
ysv<-y[(primal(sv2)<0.9999*C) & (primal(sv2)>0.0001*C)][1] #CLASE DEL 1ER S.V.
be<-w%*%xsv-ysv
```

R03: Resolución de la formulación primal del ejemplo de ilustración de una support vector machine

```
### PROBLEMA PRIMAL
##  CALCULO DE LOS ARGUMENTOS DE ipop PARA EL PROBLEMA PRIMAL
H<-rep(0,(m+t)^2)
dim(H)<-c(m+t,m+t)
diag(H)<-c(rep(1,t-1),rep(0,m+1))
aux<-as.vector(x)*as.vector(y)
dim(aux)<-dim(x)
A<-cbind(aux,-y,diag(m))
c<-c(rep(0,t),rep(C,m))
```

```
b<-rep(1,m)
r<-rep(1000000,m)
l<-c(rep(-1000000,t),rep(0,m))
u<-rep(1000000,m+t)

## OPTIMIZADOR CUADRÁTICO PARA EL PROBLEMA PRIMAL
sv1<-ipop(c,H,A,b,l,u,r)
sv1
```

R04: Resolución del ejemplo de ilustración de una support vector machine con kernel polinómico homogéneo de grado 2

```
library(kernlab)
### LECTURA DE LOS DATOS
datos<-read.csv(file="DatosSVM.csv",header=TRUE,sep=";",dec=",")
m<-dim(datos)[1]
t<-dim(datos)[2]
x<-as.matrix(datos[,-t])
y<-as.matrix(datos[,t])

### PROBLEMA DUAL
##  CALCULO DE LOS ARGUMENTOS DE ipop PARA EL PROBLEMA DUAL
C <- 5 #PARAMETRO DE COSTE
d<-0
g<-2
K<-(x%*%t(x)+d)^g
H<-K*(y%*%t(y))
c <- matrix(rep(-1,m))
A <- t(y)
b <- 0
l <- matrix(rep(0,m))
u <- matrix(rep(C,m))
r <- 0

##  OPTIMIZADOR CUADRÁTICO PARA EL PROBLEMA DUAL
sv2 <- ipop(c,H,A,b,l,u,r)
sv2
```

R05: Representación gráfica del ejemplo de ilustración de una support vector machine con kernel polinómico homogéneo de grado 2

```
## En este código se emplea la función ellipsePoints
## desarrollada por Martin Maechler
plot(x[,1],x[,2],pch=y+2,cex=2)
ep <- ellipsePoints(a=12.3263,b=1.6624,alpha=45,loc=c(0,0))
ep2 <- ellipsePoints(a=5.6559,b=0.7628,alpha=45,loc=c(0,0))
ep3 <- ellipsePoints(a=16.4891,b=2.2238,alpha=45,loc=c(0,0))
polygon(ep, col = 0)
polygon(ep2, col = 0,lty=2)
polygon(ep3, col = 0,lty=2)
```

R06: Ejemplo de ilustración de un análisis de componentes principales a partir de la matriz de covarianzas y a partir de la matriz de productos escalares

```
### LECTURA DE DATOS

datos<-read.csv(file="DatosKernelPCA.csv",header=TRUE,sep=";",dec=",")
l<-dim(datos)[1]

### PCA LINEAL
### CON DOS MÉTODOS
```

```
### 1) A PARTIR DE LA MATRIZ t(X)%*%X
### 2) A PARTIR DE LA MATRIZ DE PRODUCTOS ESCALARES

##  CENTRADO DE DATOS
X<-scale(datos,center=TRUE,scale=FALSE)

##  MATRIZ t(X)%*%X
C<-t(X)%*%X # LA MATRIZ DE COVARIANZAS ES, EN REALIDAD 1/l * C

##  MATRIZ DE PRODUCTOS ESCALARES
K<-X%*%t(X)

##  DESCOMPOSICION ESPECTRAL DE LA MATRIZ t(X)%*%X
LAMBDA_C<-eigen(C)$values
W_C<-eigen(C)$vectors

##  DESCOMPOSICION ESPECTRAL DE LA MATRIZ DE PRODUCTOS ESCALARES
LAMBDA_K<-eigen(K)$values
W_K<-eigen(K)$vectors

##  PROYECCIONES DE LOS INDIVIDUOS EN LOS PC A PARTIR DE LA MATRIZ t(X)%*%X
F_C<-X%*%W_C

##  PROYECCIONES DE LOS INDIVIDUOS EN LOS PC A PARTIR DE LA MATRIZ DE PRODUCTOS ESCALARES
F_K<-K%*%W_K%*%diag(LAMBDA_K^(-1/2))
```

R07: Ejemplo de ilustración del kernel PCA sin kernel trick

```
### KERNEL PCA SIN KERNEL TRICK
##
##  CALCULO DE LAS COORDENADAS DE LOS ELEMENTOS EN EL FEATURE SPACE
##  EN EL CASO DEL KERNEL POLINÓMICO DE GRADO 2 Y OFFSET 0
##  EL FEATURE SPACE TIENE DIMENSIÓN 6
##  Y LAS COORDENADAS DE LOS INDIVIDUOS EN EL FEATURE SPACE
##  SE CALCULAN DEL SIGUIENTE MODO
f1<-X[,1]^2
f2<-X[,2]^2
f3<-X[,3]^2
f4<-sqrt(2)*X[,1]*X[,2]
f5<-sqrt(2)*X[,1]*X[,3]
f6<-sqrt(2)*X[,2]*X[,3]
X_F<-cbind(f1,f2,f3,f4,f5,f6)

##  CENTRADO DE LA MATRIZ DE DATOS EN EL FEATURE SPACE
X_F_C<-scale(X_F,center=TRUE,scale=FALSE)

##  MATRIZ DE COVARIANZAS EN EL FEATURE SPACE
CF<-t(X_F_C)%*%X_F_C # NO ES EXACTAMENTE LA MATRIZ DE COVARIANZAS, SINO l*C

##  DESCOMPOSICION ESPECTRAL DE LA MATRIZ DE COVARIANZAS EN EL FEATURE SPACE
LAMBDA_CF<-eigen(CF)$values
W_CF<-eigen(CF)$vectors

### PROYECCIONES DE LOS INDIVIDUOS EN LOS PC A PARTIR DE LA MATRIZ DE COVARIANZAS EN EL FEATURE
### SPACE
F_CF<-X_F_C%*%W_CF
```

R08: Ejemplo de ilustración del kernel PCA con kernel trick

```
##  CÁLCULO INICIAL DE LA MATRIZ KERNEL
KMI<-(X%*%t(X))^2

##  CENTRADO DE LOS DATOS EN EL FEATURE SPACE
j<-rep(1,l)
dim(j)<-c(l,1)
KM<-KMI-1/l*(KMI%*%j%*%t(j))-1/l*(j%*%t(j)%*%KMI)+(1/l^2)*det(t(j)%*%KMI%*%j)*(j%*%t(j))

## DESCOMPOSICION ESPECTRAL DE LA MATRIZ KERNEL
LAMBDA_KM<-eigen(KM)$values
W_KM<-eigen(KM)$vectors
```

```
## PROYECCIONES DE LOS INDIVIDUOS EN LOS COMPONENTES PRINCIPALES
F_KM<-KM%*%W_KM%*%diag((LAMBDA_KM)^(-1/2))
```

R09: Resolución del ejemplo de ilustración de support vector machine con kernel lineal con el package e1071

```
library(e1071)
datos.svm<-read.csv(file="DatosEjemploIlustracion.csv", header=TRUE,sep=";")
datos.svm$clase<-as.factor(datos.svm$clase)
## SUPPORT VECTOR MACHINE LINEAL CON E1071 CON C=5
svm1<-svm(clase~.,data=datos.svm,kernel="linear",cost=5,scale=FALSE)
plot(svm1,datos.svm,grid=100)
svm1$SV
svm1$coefs
pred<-predict(svm1,datos.svm[,1:2])
table(pred,datos.svm[,3])
```

R10: Resolución del ejemplo de ilustración de support vector machine con kernel polinómico homogéneo de grado 2 con el package e1071

```
## SUPPORT VECTOR MACHINE POLINÓMICO HOMOGÉNEO DE GRADO 2 CON E1071 CON C=5

svm2<-svm(clase~.,data=datos.svm,kernel="polynomial",gamma=1,coef0=0,degree=2,cost=5, scale=FALSE)
plot(svm2,datos.svm,grid=100)
svm2$SV
svm2$coefs
pred<-predict(svm2,datos.svm[,1:2])
table(pred,datos.svm[,3])
```

R11: Resolución del ejemplo de ilustración de support vector machine con kernel lineal con el package kernlab

```
## SUPPORT VECTOR MACHINE LINEAL CON KERNLAB CON C=5
require(kernlab)
x <- as.matrix(datos.svm[,1:2])
y <- as.matrix(datos.svm[,3])
svp1 <- ksvm(x,y,type="C-svc",kernel="vanilladot",C=5,scaled=FALSE)
plot(svp1,data=x)
alpha(svp1)
alphaindex(svp1)
coef(svp1)
b(svp1)
nSV(svp1)
obj(svp1)
error(svp1)
pred<-predict(svp1,as.matrix(datos.svm[,1:2]))
table(pred,datos.svm[,3])
```

R12: Representación gráfica alternativa para el ejemplo de ilustración de support vector machine con kernel lineal

```
x<-seq(min(datos.svm[,1])-1,max(datos.svm[,1])+1,length.out=100)
y<-seq(min(datos.svm[,2])-1,max(datos.svm[,2])+1,length.out=100)
```

```
grid<-expand.grid(x,y)
w<-predict(svp1,grid,type="decision")
dim(w)<-c(length(x),length(y))
image(x,y,w,col=terrain.colors(96))
contour(x,y,w,levels=c(-1,0,1),add=TRUE)
supportvector<-rep(0,dim(datos.svm)[1])
supportvector[alphaindex(svp1)[[1]]]<-1
points(datos.svm[,1:2],col=datos.svm[,3],cex=supportvector*0.5+1,lwd=supportvector*0.5+1,pch=19)
```

R13: Resolución del ejemplo de ilustración de support vector machine con kernel polinómico homogéneo de grado 2 con el package kernlab

```
## SUPPORT VECTOR MACHINE POLINÓMICO HOMEGÉNEO DE GRADO 2 CON KERNLAB CON C=5
x <- as.matrix(datos.svm[,1:2])
y <- as.matrix(datos.svm[,3])

svp2 <- ksvm(x,y,type="C-
svc",kernel="polydot",C=5,kpar=list(degree=2,scale=1,offset=0),scaled=FALSE)
plot(svp2,data=x)
alpha(svp2)
alphaindex(svp2)
coef(svp2)
b(svp2)
nSV(svp2)
obj(svp2)
error(svp2)
pred<-predict(svp2,as.matrix(datos.svm[,1:2]))
table(pred,datos.svm[,3])
```

R14: Representación gráfica alternativa para el ejemplo de ilustración de support vector machine con kernel polinómico homogéneo de grado 2

```
x<-seq(min(datos.svm[,1])-1,max(datos.svm[,1])+1,length.out=100)
y<-seq(min(datos.svm[,2])-1,max(datos.svm[,2])+1,length.out=100)
grid<-expand.grid(x,y)
w<-predict(svp2,grid,type="decision")
dim(w)<-c(length(x),length(y))
image(x,y,w,col=terrain.colors(96))
contour(x,y,w,levels=c(-1,0,1),add=TRUE)
supportvector<-rep(0,dim(datos.svm)[1])
supportvector[alphaindex(svp2)[[1]]]<-1
points(datos.svm[,1:2],col=datos.svm[,3],cex=supportvector*0.5+1,lwd=supportvector*0.5+1,pch=19)
```

R15: Creación de la base de datos para la support vector machine sobre variables de competitividad en las regiones europeas

```
library(FactoMineR)
library(e1071)
library(kernlab)

##  LECTURA DE DATOS
tabla<-read.csv(file="DatosDefinitivos.csv", header=TRUE)
datos<-tabla[,-c(1,2)]
datos$Dens<-log(datos$Dens)
attach(datos)
rownames(datos)<-tabla[,2]

##  ANÁLISIS DE COMPONENTES PRINCIPALES INICIAL
acp1<-PCA(datos,ncp=20,quali.sup=c(1,2),quanti.sup=c(3,4),row.w=datos$POP)
clase<-rep(-1,188)
clase[UE15=="UE15"]<-1
```

```
##  CREACIÓN DE LA TABLA DE DATOS COMO INPUT DEL SVM
datos.svm<-as.data.frame(cbind(acp1$ind$coord[,1:2],as.factor(clase)))
datos.svm$V3<-as.factor(datos.svm$V3)
```

R16: Tuning del kernel gaussiano

```
## TUNING DEL KERNEL RADIAL
obj <- tune(svm, V3~., data = datos.svm,
ranges = list(gamma = 2^(-2:2), cost = 2^(0:4)),
tunecontrol = tune.control(sampling = "fix"))
obj
```

R17: Ejemplo de support vector machine sobre variables de competitividad de regiones europeas con el package e1071

```
x<-1:188
sel<-sample(x)
datostraining<-datos.svm[sel[1:125],] ## DOS TERCIOS DE MUESTRA DE APRENDIZAJE
datostest<-datos.svm[sel[126:188],] ## UN TERCIO DE MUESTRA TEST
svm2<-svm(V3~.,data=datostraining,kernel="radial",gamma=0.25,cost=1)
plot(svm2,datostraining,grid=100)
predtraining<-predict(svm2,datostraining[,1:2])
table(predtraining,datostraining[,3])
plot(svm2,datostest,grid=100)
predtest<-predict(svm2,datostest[,1:2])
table(predtest,datostest[,3])
```

R18: Ejemplo de support vector machine sobre variables de competitividad de regiones europeas con el package kernlab

```
## SUPPORT VECTOR MACHINE CON KERNLAB (MUESTRA DE APRENDIZAJE Y MUESTRA TEST)
x<-1:188
sel<-sample(x)
xtraining<-as.matrix(datos.svm[sel[1:125],1:2])
ytraining<-as.matrix(datos.svm[sel[1:125],3])
xtest<-as.matrix(datos.svm[sel[126:188],1:2])
ytest<-as.matrix(datos.svm[sel[126:188],3])
svp2 <- ksvm(xtraining,ytraining,type="C-svc",kernel="rbfdot",C=1,kpar=list(sigma=0.25))
dev.new()
plot(svp2,data=xtraining)
dev.new()
plot(svp2,data=xtest)
predtraining<-predict(svp2,as.matrix(xtraining))
table(predtraining,ytraining)
predtest<-predict(svp2,as.matrix(xtest))
table(predtest,ytest)
```

R19: Representación gráfica alternativa para el ejemplo de support vector machine sobre variables de competitividad en regiones europeas

```
x<-seq(min(xtraining[,1])-1,max(xtraining[,1])+1,length.out=100)
y<-seq(min(xtraining[,2])-1,max(xtraining[,2])+1,length.out=100)
grid<-expand.grid(x,y)
```

```
w<-predict(svp2,grid,type="decision")
dim(w)<-c(length(x),length(y))

## GRÁFICO PARA LA MUESTRA DE APRENDIZAJE
dev.new()
image(x,y,w,col=terrain.colors(96),main="Muestra de aprendizaje")
contour(x,y,w,levels=c(-1,0,1),add=TRUE)
supportvector<-rep(0,dim(xtraining)[1])
supportvector[alphaindex(svp2)[[1]]]<-1
points(xtraining[,1:2],col=ytraining,cex=supportvector*0.5+1,lwd=supportvector*0.5+1,pch=19)


## GRÁFICO PARA LA MUESTRA TEST
dev.new()
image(x,y,w,col=terrain.colors(96),main="Muestra test")
contour(x,y,w,levels=c(0),add=TRUE)
points(xtest[,1:2],col=ytest,pch=19)
```

R20: Lectura de la base de datos para el ejemplo artificial de kernel PCA

```
datos<-read.csv(file="ToyExampleData.csv",header=TRUE,sep=";",dec=",")
datos$clase<-as.factor(datos$clase)
dim(datos)
summary(datos)
```

R21: Representación de los elementos en el input space con escala de grises para el ejemplo artificial de kernel PCA (primer componente principal)

```
x<-seq(min(datos[,1]),max(datos[,1]),length.out=100)
y<-seq(min(datos[,2]),max(datos[,2]),length.out=100)
grid<-expand.grid(x,y)
colnames(grid)<-c("X","Y")
pred<-predict(kpca2,grid)
valor<-pred[,1]
dim(valor)<-c(length(x),length(y))
image(x,y,valor,col=gray.colors(96))
contour(x,y,valor,add=TRUE)
points(datos[,1:2],pch=as.integer(datos[,3]))
```

R22: Representación de los elementos en el input space con escala de grises para el ejemplo artificial de kernel PCA (segundo componente principal)

```
x<-seq(min(datos[,1]),max(datos[,1]),length.out=100)
y<-seq(min(datos[,2]),max(datos[,2]),length.out=100)
grid<-expand.grid(x,y)
colnames(grid)<-c("X","Y")
pred<-predict(kpca2,grid)
valor<-pred[,1]
dim(valor)<-c(length(x),length(y))
image(x,y,valor,col=gray.colors(96))
contour(x,y,valor,add=TRUE)
points(datos[,1:2],pch=as.integer(datos[,3]))
```

R23: Cálculo de la matriz de nuevas distancias a partir de la matriz kernel en el ejemplo artificial de kernel PCA

```
##   CÁLCULO DE LA MATRIZ KERNEL
dt<-as.matrix((datos[,1:2]))
rbf<-rbfdot(sigma=0.00722)
KERNEL<-kernelMatrix(rbf,dt)

##   CÁLCULO DE LA MATRIZ DE DISTANCIAS DERIVADA DE LA MATRIZ KERNEL
DIST<-rep(0,dim(KERNEL)[1]^2)
dim(DIST)<-dim(KERNEL)
for(i in 1:dim(KERNEL)[1])
  {
  for(j in 1:dim(KERNEL)[2])
    {
    DIST[i,j]=(KERNEL[i,i]+KERNEL[j,j]-2*KERNEL[i,j])^(1/2)
    }
  }
```

R24: Escalamiento multidimensional a partir de la matriz de nuevas distancias en el ejemplo artificial de kernel PCA

```
##   MULTIDIMENSIONAL SCALING EN DOS DIMENSIONES A PARTIR DE LA MATRIZ DE DISTANCIAS
fit <- cmdscale(DIST,eig=TRUE, k=2)
plot(fit$points,pch=as.integer(datos[,3]))
```

R25: Lectura de la base de datos para el ejemplo de kernel PCA de variables de competitividad en regiones europeas

```
datos<-read.csv(file="ExampleKernelPCA.csv",header=TRUE)
datos$UE15<-as.factor(datos$UE15)
datos<-datos[,-4]
datos[,2:3]<-scale(datos[,2:3])
dim(datos)
summary(datos)
```

R26: Representación de los elementos en el input space con escala de grises para el ejemplo de kernel PCA sobre variables de competitividad en regiones europeas (primer componente principal)

```
## DIBUJO CON ESCALA DE GRISES CON VALORES DE LOS COMPONENTES PRINCIPALES NO LINEALES

x<-seq(min(datos[,2]),max(datos[,2]),length.out=100)
y<-seq(min(datos[,3]),max(datos[,3]),length.out=100)
grid<-expand.grid(x,y)


colnames(grid)<-colnames(datos)[c(2,3)]
pred<-predict(kpca2,grid)
valor<-pred[,1]
dim(valor)<-c(length(x),length(y))
image(x,y,valor,col=gray.colors(96))
contour(x,y,valor,add=TRUE)
points(datos[,2:3],pch=as.integer(datos[,1]))
```

R27: Representación de los elementos en el input space con escala de grises para el ejemplo de kernel PCA sobre variables de competitividad en regiones europeas (segundo componente principal)

```
x<-seq(min(datos[,2]),max(datos[,2]),length.out=100)
y<-seq(min(datos[,3]),max(datos[,3]),length.out=100)
grid<-expand.grid(x,y)
colnames(grid)<-colnames(datos)[c(2,3)]
pred<-predict(kpca2,grid)
valor<-pred[,2]
dim(valor)<-c(length(x),length(y))
image(x,y,valor,col=gray.colors(96))
contour(x,y,valor,add=TRUE)
points(datos[,2:3],pch=as.integer(datos[,1]))
```

R28: Cálculo de nuevas distancias a partir de la matriz kernel y escalamiento multidimensional para el ejemplo de kernel PCA sobre variables de competitividad en regiones europeas

```
##   CÁLCULO DE LA MATRIZ KERNEL
dt<-as.matrix((datos[,1:2]))
rbf<-rbfdot(sigma=0.00722)
KERNEL<-kernelMatrix(rbf,dt)

##   CÁLCULO DE LA MATRIZ DE DISTANCIAS DERIVADA DE LA MATRIZ KERNEL
DIST<-rep(0,dim(KERNEL)[1]^2)
dim(DIST)<-dim(KERNEL)
for(i in 1:dim(KERNEL)[1])
{
for(j in 1:dim(KERNEL)[2])
{
DIST[i,j]=(KERNEL[i,i]+KERNEL[j,j]-2*KERNEL[i,j])^(1/2)
}
}

##   MULTIDIMENSIONAL SCALING EN DOS DIMENSIONES A PARTIR DE LA MATRIZ DE DISTANCIAS
fit <- cmdscale(DIST,eig=TRUE, k=2)
plot(fit$points,pch=as.integer(datos[,3]))
```