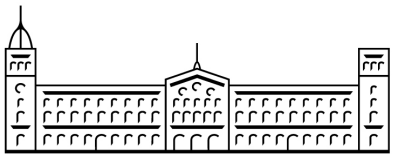

Treball final de grau

GRAU DE MATEMÀTIQUES

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

ANÀLISI DE COMPONENTS PRINCIPALS AMB KERNELS

Autor: Núria Vilches Baldonado

Director:	**Dr. Esteban Vegas Lozano**
Realitzat a:	**Departament de Genètica, Microbiologia i Estadística**
Barcelona,	**17 de gener de 2017**

Índex

Índex de figures

Notació

x	valor escalar
$\mathbf{x}$	vector columna
$\mathbf{X}$	matriu
m	nombre d'observacions o individus
n	nombre de variables
$\mathcal{X}$	espai d'entrada o *input space*
$\mathcal{F}$	espai de característiques o *feature space*
$k(\mathbf{x}, \mathbf{y})$	funció kernel
$\mathbf{K}$	matriu kernel
$\mathbf{I}$	matriu identitat
$\mathbf{1}_m$	vector d'uns de dimensió m

1 Introducció

En els darrers anys hi ha hagut un augment en l'interès de l'anàlisi de dades però alhora una crescuda en la quantitat de dades a emmagatzemar. És per això, que han començat a aparèixer nous algorismes d'anàlisi de dades més òptims i eficients que els anteriors que faciliten l'anàlisi de gran quantitat de dades, l'anomenat *Big Data*. Moltes d'aquestes metodologies es troben dintre de l'anomenat *machine learning* o aprenentatge automàtic.

Tots aquests mètodes són cada cop més aplicats en el dia a dia de les persones, en l'actualitat la majoria de grans empreses, com bancs, botigues, pàgines web, utilitzen l'estudi de dades en el desenvolupament de campanyes de publicitat, venta personalitzada, entre d'altres.

Un altra forma d'aplicar aquests mètodes és en l'agrupació de les dades. Per exemple, en l'establiment de venta al públic anterior, es poden intentar agrupar les persones segons els tipus de productes que compren o altres propietats (edat, sexe, etc). D'aquesta forma, quan arribi un nou client, podem classificar-lo en un d'aquest grups i vendre els productes que acostumen a comprar les persones classificades en el mateix grup.

No és difícil d'entendre l'abast de l'anàlisi del *Big Data* en el món actual. Si som capaços d'analitzar un conjunt de dades fins a poder predir els esdeveniments amb mínim error, tindrem en les nostres mans la possibilitat de invertir en el futur amb un risc mínim.

Aquest tema m'interessa especialment des de que vaig escollir la menció en estadística. Allà les assignatures que vaig cursar hem van motivar a fer un treball de recerca en estadística i, especialment en *machine learning*, és per això que quan s'hem va presentar l'oportunitat no vaig dubtar en agafar-la. A més a més, la meva experiència en una consultoria dedicada al *Big Data* hem va donar la oportunitat de veure de primera mà la quantitat d'aplicacions de l'anàlisi de dades i, en particular de la importància de una clara visualització de les dades per facilitat el dia a dia dels treballadors.

En el departament d'Estadística de la Universitat de Barcelona s'està investigant àmpliament aquests mètodes, entre els que trobem l'estudi del kernel PCA, on busquen afrontar la manca d'interpretabilitat d'aquest i explorar el seu ús en la integració de dades.

En particular, al 2015 una estudiant de màster d'estadística de la Universitat de Barcelona va iniciar la investigació de la millora de la interpretació del mètode kernel PCA fins a implementar un paquet en llenguatge R que representa el kernel PCA d'una forma més clara. Aquest treball és una continuació del que l'alumna Núria Planell va fer, ampliant la metodologia del kernel PCA i, sobretot, ampliant el paquet en R per que pugui arribar a un conjunt de dades més variat.

Però es clar que encara que tinguem un mètode molt òptim i sense costos computacionals, si no podem interpretar els resultats, equival a no tenir cap mètode. Una cosa semblant succeeix amb el mètode de kernel PCA. En aquest treball es pretén

millorar la interpretació del mètode kernel PCA, projectant les variables originals en el gràfic del kernel PCA, recuperant així part de la informació que ens donen les dades originals.

El principal objectiu del projecte és arribar a una varietat de dades més gran desenvolupant noves funcions kernel en la interpretació del kernel PCA. Per fer-ho es farà una revisió de les funcions kernel i l'anàlisi de components principals basat en funcions kernel. Amb especial èmfasi en la integració de dades i en les noves metodologies per a la visualització de variables. A més a més, s'ampliarà la funció en llenguatge R, implementada per la Núria Planell, facilitant la integració de variables originals per a les diferents funcions kernel.

La metodologia de la recerca és:

1. Recerca prèvia sobre els aspectes bàsica dels mètode, on s'inclou informació sobre el *machine learning*.
2. Definició dels mètodes basats en kernel i exemples de les seves aplicacions.
3. Definició de l'anàlisi de components principals per a funcions kernel.
4. Algorisme de la representació de variables originals.
5. Definició de diferents funcions kernel i càlcul de la projecció de variables originals segons aquests kernels.
6. Desenvolupament del paquet estadístic en R per a implementar els càlculs anteriors.

2 Machine Learning

El *machine learning* (o aprenentatge estadístic) és una disciplina dins de la intel·ligència artificial que busca algoritmes dins de les dades per intentar predir comportaments futurs i, a mesura que les dades evolucionen, l'algorisme ho fa amb elles.

És a dir, amb el *machine learning* el que es pretén és trobar relacions o patrons en conjunts de dades per poder aplicar un mètode de predicció.

Segons Michell [15], el *machine learning* el que pretén és respondre la següent pregunta:

Cóm podem construir sistemes que automàticament milloren amb l'experiència, i quines són les lleis fonamentals que governen qualsevol procès d'aprenentatge?

Aquest tipus d'algorisme s'utilitzen en processos en els que és molt costós trobar l'algorisme o per processos on hi juga l'atzar. Per exemple, podem aplicar aquest mètode per diferenciar els emails legítims del spam. En aquest cas sabem que les dades d'entrada són un conjunt de paraules dins d'un email, la variable de resposta és sí o no referint-nos a si és un email d'spam o legítim, però desconeixem cóm arribar d'un punt a l'altre. D'aquesta forma podem compilar uns mil·lers d'exemples de correu que coneixem com a spam i crear un algorisme que aprengui les característiques d'aquests correus, donant-nos així l'algorisme que buscàvem en un principi i que diferència entre correu legítim i spam.

Les aplicacions del *machine learning* són molt diverses, a continuació es mostren alguns dels àmbits en els que es pot aplicar, però n'existeixen molts més:

- En les compres online, és una forma de poder destinar a cada usuari una publicitat personalitzada segons els seus gustos
- En la política, s'utilitza per crear una campanya adequada segons les publicacions dels votants i per saber quin és el millor moment per publicar tuits o actualitzacions de facebook.
- En algunes empreses el fan servir per coses tan diverses com detectar frau en transaccions, predir errors en equips tècnics, predir quins seran els empleats més rentables al següent any o bé seleccionar clients potencials a partir de comportaments en les xarxes socials.
- En la medicina es pot aplicar per trobar molècules que formin nous medicaments o per relacionar gens amb malalties.

Per ser més precisos, una màquina aprèn respecte a una tasca concreta si a mesura que adquireix l'experiència, augmenta el rendiment de la tasca.

2.1 Avantatges i desavantatges del *machine learning*

Com qualsevol mètode estadístic, el *machine learning* té les seus inconvenients i els seus avantatges, anem a mirar alguns d'aquests. Les seves principals avantatges són:

- Facilita la presa de decisions.
- Accelera els processos.
- Automatitza els processos.
- No necessita de l'experiència humana.
- S'aplica en tasques que només es poden definir a partir d'exemples.
- Troba les relacions i correlacions entre gran quantitat de dades.
- Evoluciona amb les dades.

Els desavantatges de fer servir aquest mètode són en els casos de que la màquina prengui decisions importants de forma automàtica i no supervisada que haurien d'involucrar a les persones, en aquests casos existeix el risc de que la màquina prengui decisions errònies. Per una altra banda, també pot ocórrer que l'enginyer faci una mala implementació de l'algorisme d'aprenentatge, de tal forma que es prenguin decisions a partir de pronòstics dolents.

2.2 Estadística i *machine learning*

No hi ha una línia que divideixi l'estadística i el *machine learning*, és una línia contínua. És clar que existeix un solapament entre els dos camps: els dos enfocaments d'anàlisi de dades són complementaris, més que no pas contradictoris. En la imatge 1, extreta de [16] podem veure una idea del que es comenta:

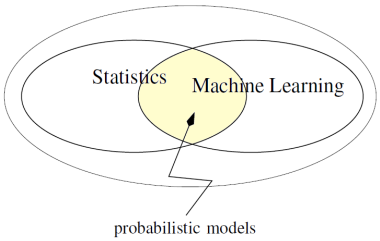

Figura 1: Estadística i *machine learning*

Els algorismes de *machine learning* tenen una base matemàtica sòlida, i molts incorporen directament estadística. Les tècniques estadístiques s'han desenvolupat de forma independent, però moltes són fonamentalment similars a l'aprenentatge

automàtic i produeixen un resultat similar. Les tècniques de validació dels models són les mateixes per a ambdós tipus d'anàlisi. Hi ha molts temes que comencen en un dels dos camps però que després es desenvolupen més en l'altre, i també hi ha temes que estan actius en les dues àrees.

Encara que el *machine learning* i l'estadística es preocupen per resoldre la mateixa pregunta: *çóm podem aprendre de les dades?"* l'enfocament principal de les dues disciplines és diferent. En la següent taula és pot veure un esquema de les diferències principals entre aquestes dues disciplines.

Estadística	*machine learning*
Inferir el procés generador de les dades	Predir com seran les futures dades respecte una variable
La mostra prové d'una població	Parteix d'unes dades exemple
Centenars o milers de dades	Mil·lons o milers de mil·lions de dades
Necessita interacció humana	Automàtica i sense supervisió
S'aplica un preprocés a les dades	Preprocés quasi innecessari
Aprèn sobre la mostra	Evoluciona amb la mostra

Una de les raons per les quals existeix una divisió entre les dues disciplines són les diferències en la terminologia: s'usen termes diferents però que es refereixen al mateix concepte. En la següent taula es mostra un petit recull de diversos termes estadístics i la seva traducció dins del món de l'aprenentatge automàtic.

Estadística	*machine learning*
Variables	Atributs (camps)
Individus	Instàncies (Registres)
Variable explicativa, predictor,...	Input
Variable resposta	Output
Model	Xarxa neuronal, arbre de decisió,...
Coeficients	Pesos
Criteri d'ajust (MV, MQO,...)	Funció de cost
Estimació	Aprenentatge
Clustering	Aprenentatge no supervisat
Classificació, Regressió,...	Aprenentatge supervisat

2.3 Tipus de tasques

Els algorismes d'aprenentatge estadístic s'organitzen segons el resultat desitjat o en el tipus d'informació d'entrada disponible durant l'aprenentatge de la màquina. Les dues classificacions principals són:

- **Aprenentatge supervisat**: l'aprenentatge supervisat consisteix en establir un mecanisme de classificació o regressió a partir d'unes dades d'entrenament amb el valor de la variable d'interès conegut. Un cop establert el mecanisme predictor, s'avalua la capacitat d'aquest en un conjunt de dades de prova.

Normalment es disposa d'un conjunt de predictors $x_1, \ldots, x_n$ mesurats en m observacions i una resposta y associada a cada observació. El propòsit és ajustar un model que relacioni la resposta amb els predictors, per millorar la comprensió de la relació entre la resposta i els predictors. Els problemes principals d'aquest tipus d'aprenentatge són de classificació (quan la resposta y és una variable qualitativa) o bé de regressió (quan y és numèrica).

- **Aprenentatge no supervisat**: es caracteritza per la manca d'una variable d'interès en el conjunt de dades. Les dades no presenten una estructura prèviament establerta i l'objectiu d'aquests tipus d'anàlisi consisteix en detectar l'existència de determinades estructures en el conjunt de dades.

  En aquest cas només es disposa d'un conjunt de variables $x_1, \ldots, x_n$ mesurades en m observacions i cap variable resposta. L'objectiu és intentar trobar característiques interessants sobre les variables. Les tècniques d'aprenentatge no supervisat més populars són les tècniques de reducció de la dimensió (PCA (*Principal Components Analysis*), MSD (*Multidimensional scaling*),...) i els mètodes de clusterització de dades (*Clustering Methods*).

La varietat de les dades amb les que ens podem trobar en l'estudi ha portat al desenvolupament de mètodes més versàtils que permetin processar, analitzar i comparar molts tipus de dades diferents, com els mètodes basats en Kernels.

3 Mètodes basats en Kernel

Tradicionalment, els algorismes del *machine learning* i l'estadística han sigut suficients per predir els casos lineals, però els problemes d'anàlisis del món real molts cops necessiten mètodes no lineals per detectar el tipus de dependència que permet una predicció òptima de les variables d'interès.

Actualment l'ús dels mètodes Kernels en l'aprenentatge estadístic ha augmentat considerablement. La raó principal és que els Kernels permeten assignar les dades dins d'un espai de característiques de dimensió més alta, de tal forma que augmenten l'eficàcia o la potència computacional dels processos lineals. A més a més, és una manera d'estendre les hipòtesis lineals a no lineals implícitament.

3.1 Una mica d'història

Els mètodes Kernels han començat a formar part del machine Learning i de la inferència empírica recentment, però la història dels mètodes que apliquen Kernels definits positius es remunta a unes dècades, segons l'article [14]. Aronszajn (1950) i Parzen (1962) van ser uns dels primers en fer servir aquests mètodes en l'estadística. A partir d'aquí, Aizerman *et al.* (1964) va aplicar kernels definits positivament d'una forma molt semblant al que coneixem com el *kernel Trick* en l'actualitat.

El primer pas de tots aquests matemàtics va ser definir l'espai de Hilbert on l'avaluació dels punts és una funció lineal contínua. Aquest espai va ser introduït per primer cop per Stanislaw Zaremba al 1907 al intentar resoldre el valor del límit de funcions harmòniques. Aquest camp va quedar parat algun temps, fins que al 1950 Nachman Aronszajn ho van aplicar en la demostració del classificador per reduir la convergència. Per fer-ho, van haver de demostrar que un kernel definit positivament és igual a un producte escalar en un altre espai (espai de Hilbert o *feature space*) on el seu algorisme es redueix a l'algorisme de perceptron. Encara així, ells no van fer servir el *feature space* per dissenyar nous algorismes.

L'última investigació en aquest camp va ser 30 anys després, per Boser *et al.* (1992), per construir el SVM (Support Vector Machine), una generalització de l'algorisme del *hiperplà òptim*. En un inici es pensava que el gran avantatge del SVM comparat amb el algorisme del hiperplà òptim era que permeten l'ús d'una classe major de mesures de similitud. Però va ser Schölkopf (1997) qui va notar que els kernels no només augmenten la flexibilitat sinó que a més a més fan possible treballar amb dades no vectorials i els kernels donen automàticament una representació vectorial de les dades en el *feature space*.

Els primers exemples de kernels no trivials en dades no vectorials van ser desenvolupats per Haussler (1999) i Watkins (2000). A més a més, van resoldre que els kernels es poden fer servir per construir generalitzacions de qualsevol algorisme que es pugui definir en termes de producte escalar, i en els últims 5 anys s'ha aplicat els mètodes basats en kernel en un gran número d'algorismes.

3.2 Idea general

Suposem que es té un conjunt $S = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$ de m observacions que es volen analitzar, on cada objecte $\mathbf{x}_\alpha$ és un element del conjunt $\mathcal{X}$ anomenat espai d'entrada o *input space*. Aleshores definim un espai de característiques o *feature space* $\mathcal{F}$, on aplicarem les tècniques estadístiques lineals. Aquest espai de característiques és de dimensió més elevada que $\mathcal{X}$ (pot ser infinita) i es crea a partir de una transformació definida per l'aplicació ϕ:

$$\begin{array}{rccc} \phi: & \mathcal{X} & \longrightarrow & \mathcal{F} \\ & \mathbf{x} & \longrightarrow & \phi(\mathbf{x}) \end{array}$$

on ϕ és un mapeig no lineal. Això fa que en l'espai de característiques $\mathcal{F}$ es puguin aplicar tècniques lineals a les dades transformades $\phi(\mathcal{X})$ que en l'espai d'entrada equivalen a tècniques no lineals.

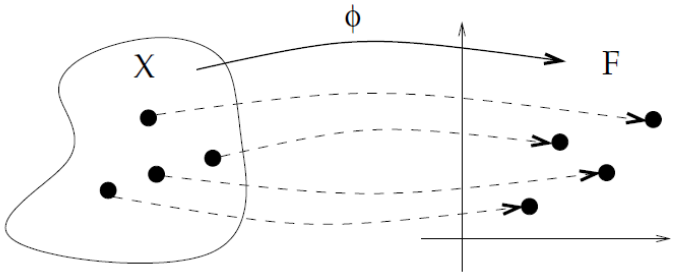

Figura 2: Transformació de les dades originals a l'espai de característiques

El problema ve a l'hora de calcular $\phi(\mathbf{x})$ en el nou espai, ja que les projeccions són molt costoses quan la dimensió de $\mathcal{F}$ és molt gran. És per això que en comptes de calcular aquesta funció, calculem el seu producte escalar en l'espai de característiques. Aquesta funció s'anomena **funció kernel** i es defineix com:

$$\begin{array}{rccc} k: & \mathcal{X} \times \mathcal{X} & \longrightarrow & \mathbb{R} \\ & (\mathbf{x}, \mathbf{y}) & \longrightarrow & \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle \end{array}$$

D'aquesta forma, ens estalviem el càlcul de les coordenades de les dades en el *feature space* i, fins hi tot, de la funció de transformació ϕ. L'únic inconvenient d'utilitzar la funció kernel és que només s'apliquen en aquelles tècniques estadístiques en que intervenen productes escalars.

El gran avantatge de l'ús dels kernels, per contra, es pot veure en la figura 3, on veiem que en l'espai original $\mathcal{X}$ els punts no són linealment separables, però en l'espai de característiques $\mathcal{F}$ obtenim unes dades transformades separables linealment. D'aquesta forma, en el nou espai podem aplicar mètodes lineals que donin resultats no lineals en l'espai original.

Per tant, entenem com un mètode de kernel aquells per als quals les dades a ser analitzades només entren en l'algorisme mitjançant la funció kernel. El resultat que

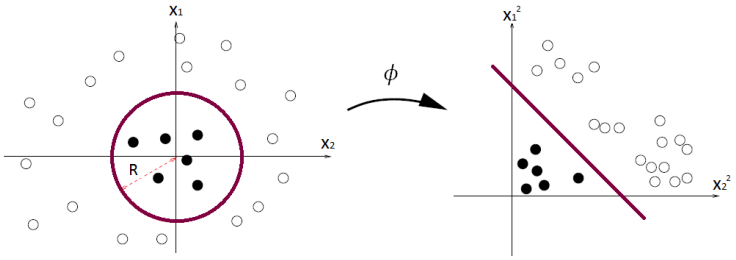

Figura 3: Separació de variables amb una relació no lineal

dóna lloc a aquesta afirmació es coneix com el kernel *trick*. Hi ha molts algorismes que poden usar kernels, com per exemple les màquines de vector suport (SVM), el *clustering*, l'anàlisi de components principals (PCA), l'anàlisi de correlacions canòniques (CCA), la *ridge regression*, etc.

Kernel Trick

El kernel *trick* és el concepte d'entendre els kernels com a productes escalars. La idea és que qualsevol algorisme per a dades vectorials que es pugui expressar com a producte escalar es pot realitzar implícitament en el *feature space* associat a qualsevol kernel, mitjançant la substitució de cada producte escalar pel resultat del kernel.

Proposició 3.1. *Qualsevol algorisme per dades vectorials que es pugui expressar únicament en termes de producte escalar entre vectors, es pot calcular implícitament en l'espai de característiques associat a partir d'un kernel, reemplaçant cada producte escalar per una funció kernel.*

Es clar que el kernel *trick* té una gran quantitat d'aplicacions, però s'han començat a desenvolupar recentment. En primer lloc, és una metodologia que permet transformar algorismes lineals (com l'anàlisis de components principals (PCA)) en algorismes no lineals, simplement per reemplaçar el producte escalar euclidi per un kernel general. Per tant, la no linealitat es obtinguda sense cost computacional, ja que l'algorisme es manté exactament igual.

En segon lloc, la combinació del kernel *trick* amb funcions kernel definides en dades no vectorials permet l'aplicació de molts algorismes clàssics en vectors de quasi qualsevol tipus de dades.

3.3 Mètodes basats en Kernel

Començarem per definir la funció de Kernel formalment.

Definició 3.1.1. *Partint d'un conjunt de dades $\mathcal{X}$ qualsevol de naturalesa indiferent (vectors, textos, imatges, etc.), una* ***aplicació Kernel*** *és una funció $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ tal que*

1. *és simètrica*

$$k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$$

2. *definida positiva*

$$\sum_{i,j=1}^{m} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

per a qualsevol $m \in \mathbb{N}, \alpha_i \in \mathbb{R}, \mathbf{x}_i \in \mathcal{X}$, $\forall i = 1, \ldots, m$.

La funció Kernel pot ser interpretada com a una mesura de similitud entre les mostres i ens permet identificar cada valor $\mathbf{x} \in \mathcal{X}$ amb un valor real que conté tota la informació rellevant a partir de l'aplicació d'*embedding*.

Definició 3.1.2. *L'**aplicació embedding** es defineix per*

$$\begin{array}{rccc} \phi\colon & \mathcal{X} & \longrightarrow & \mathcal{F} = \{f | f : \mathcal{X} \to \mathbb{R}\} \\ & \mathbf{x} & \longmapsto & \phi(\mathbf{x}) \equiv k(\cdot, \mathbf{x}) \end{array}$$

*Aquesta funció projecta els elements de $\mathcal{X}$ en un espai $\mathcal{F}$, anomenat espai de **característsitques o feature space**. Anem a estudiar les propietats d'aquest espai.*

Podem considerar que $\phi(X) \subset \mathbb{R}^{\mathcal{X}}$ té l'estructura d'un espai vectorial real, que dependrà de la funció Kernel escollida. Aleshores podem construir un espai de vectors $\mathcal{P}_k \equiv span(\phi(\mathcal{X}))$, on els seus elements seran funcions de la forma

$$f(\cdot) = \sum_{i=1}^{m} \alpha_i k(\cdot, \mathbf{x}_i)$$

per $m \in \mathbb{N}$ i $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{X}, \alpha_1, \ldots \alpha_m \in \mathbb{R}$.

A més a més, la funció de Kernel ens permet definir un producte en $\mathcal{P}_k$

Definició 3.1.3. *Siguin $f, g \in \mathcal{P}_k$ de la forma $f(\cdot) = \sum_{i=1}^{m} \alpha_i k(\cdot, \mathbf{x}_i)$ i $g(\cdot) = \sum_{j=1}^{n} \beta_j k(\cdot, \mathbf{y}_j)$, aleshores es defineix el **producte escalar** entre f i g com*

$$\langle f, g \rangle \equiv \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{y}_j)$$

El qual satisfà que $\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = \langle k(\cdot, \mathbf{x}), k(\cdot, \mathbf{y}) \rangle = k(\mathbf{x}, \mathbf{y})$

A partir d'aquesta definició, podem transformar el nostre espai vectorial $\mathcal{P}_k$ en un espai de Hilbert $\mathcal{F}_k$. Aquest espai dependrà de la funció de Kernel i equival a un espai mètric de distància d definida per

$$d^2_{\mathcal{F}_k}(\phi(\mathbf{x}), \phi(\mathbf{y})) = k(\mathbf{x}, \mathbf{x}) + k(\mathbf{y}, \mathbf{y}) - 2k(\mathbf{x}, \mathbf{y})$$

Observació 3.2. Aquesta distància reflexa les diferències entre les observacions i és una bona eina per comparar-les.

A partir d'aquest moment anem a suposar que $\mathcal{X}$ és una varietat diferenciable en $\mathbb{R}^n$.

Considerem que l'aplicació Kernel és com a mínim $\mathcal{C}^2$ per als dos arguments. Aleshores una expansió de segon ordre de l'aplicació

$$h_{\mathbf{x}}(\mathbf{z}) \equiv d^2(\phi(\mathbf{x}), \phi(\mathbf{x}+\mathbf{z}) = k(\mathbf{x},\mathbf{x}) + k(\mathbf{x}+\mathbf{z},\mathbf{x}+\mathbf{z}) - 2k(\mathbf{x},\mathbf{x}+\mathbf{z}) \qquad (1)$$

indueix a una estructura Riemanniana en $\phi(\mathcal{X})$.

Definició 3.2.1. *Les components del* ***tensor mètric*** *en el espai tangent $T_{\phi(\mathbf{x})}\phi(\mathcal{X})$, sota les coordenades $\mathbf{x} = (x^1, \ldots, x^n)$ són:*

$$g_{i,j}(\mathbf{x}) = \tfrac{1}{2} D_{i,j} h_{\mathbf{x}}(\mathbf{0}) \quad i,j = 1,\ldots,n \qquad (2)$$

on $D_{i,j}$ representa la segona derivada parcial respecte les components i, j.

Matriu Kernel

La matriu kernel $\mathbf{K}$ és el resultat directe d'aplicar la funció Kernel en el conjunt de dades d'entrada $\mathcal{X}$.

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & \ldots & k(\mathbf{x}_1,\mathbf{x}_m) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_m,\mathbf{x}_1) & \ldots & k(\mathbf{x}_m,\mathbf{x}_m) \end{pmatrix}$$

Aquesta matriu és una matriu de similituds entre els individus de $\mathcal{X}$ i depèn de la funció Kernel k emprada. Es pot definir com el producte escalar en $\mathcal{F}$ per cada individu.

La matriu Kernel és el punt inicial per qualsevol mètode on s'apliquen les funcions Kernel, ja que conté tota la informació necessària. Pel seu càlcul es pot fer servir la funció `kernelMatrix` implementada en el paquet `kernlab`. Aquesta funció demana les dades a analitzar, el tipus de kernel i, en el cas de tenir paràmetres addicionals s'han d'afegir i el resultat és la matriu kernel.

En la figura 4 podem veure un esquema dels mètodes basats en kernels per a l'anàlisi de reconeixement de patrons. A partir d'un conjunt de dades es calcula la matriu kernel segons la funció kernel escollida i aleshores sobre aquesta matriu es podran aplicar els diferents algorismes o mètodes, donant-nos així la funció resultant per la detecció de patrons.

Un cop és clar el funcionament bàsic dels mètodes basats en Kernel, és fàcil observar que existeix una gran desavantatge en l'ús d'aquests mètodes en comparació amb els altres. Si ens fixem en la imatge 3 del apartat de la idea general dels mètodes basats en kernel, en l'espai de característiques hem perdut completament la informació que teníem de les dades en l'espai $\mathcal{X}$, és a dir, és molt difícil poder donar una interpretació de l'algorisme en funció de les variables originals més enllà de la seva

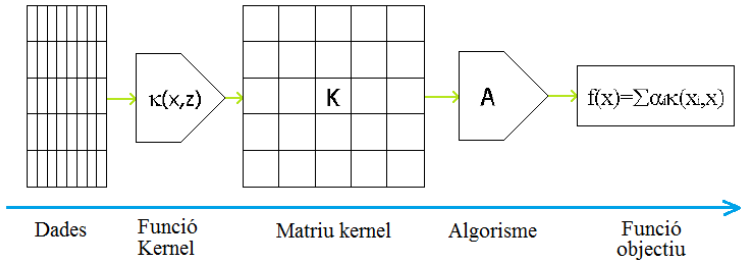

Figura 4: Esquematització dels mètodes basats en kernel

separació. És per això que en els següents capítols donarem un seguit de pautes per intentar millorar aquesta interpretació.

3.4 Tipus de Kernels

A continuació s'introdueixen els principals tipus de funcions Kernel.

Kernels per a vectors

Es tracta de funcions $k(\mathbf{x}, \mathbf{y})$ on $\mathbf{x}$ són vectors de dades. En aquest treball estudiarem amb més deteniment aquest tipus de Kernel i es poden trobar exemples en el capítol 6. Cada funció Kernel té la seva qualitat, i la tria d'una o altra depèn de les dades a analitzar i dels objectius d'anàlisi. No hi ha cap regla general per l'elecció del Kernel, sinó que s'ha d'anar provant fins a trobar el que millor ajusta les dades.

Kernels per a strings

Són funcions que comparen o compten seqüències de lletres i permeten estendre els mètodes de classificació i discriminació dels vectors numèrics a les cadenes de lletres. De forma intuïtiva, els *string* kernels són funcions que mesuren la similitud entre parelles de cadenes. Una manera de mesurar la similitud és comptant quantes subcadenes comuns d'una certa longitud tenen les dues cadenes a comparar. Els diferents Kernels que existeixen es diferencien per com valoren les coincidències entre les cadenes, alguns compten les coincidències exactes, altres permeten espais, etc. Un cop obtenim els comptatges, es deriva un vector al *feature space* i el kernel es defineix com el producte escalar entre aquests vectors. A continuació podem veure un exemple d'aquest pas de cadena de caràcters a vector:

Aquest tipus de funcions kernel són molt utilitzades en la biologia per l'anàlisi de gens.

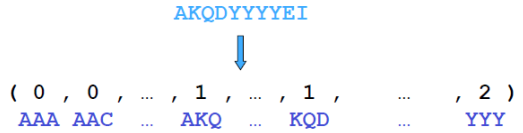

Figura 5: Exemple del càlcul d'un vector de caràcters

Combinació de Kernels

Una característica molt interessant de les funcions kernels és la combinació de kernels. Com hem vist en el apartat anterior, els kernels són funcions simètriques i definides positives i es pot demostrar que la combinació de kernels també conserven aquestes propietats. Les operacions que es podran aplicar per obtenir nous kernels vàlids que mantinguin la simetria i que es defineixin positivament són:

$$k(\mathbf{x},\mathbf{y}) = \begin{cases} k_1(\mathbf{x},\mathbf{y}) + k_2(\mathbf{x},\mathbf{y}) \\ k_1(\mathbf{x},\mathbf{y}) + k_2(\mathbf{x},\mathbf{y}) \\ f(\mathbf{x})f(\mathbf{y}) \\ k_3(\phi(\mathbf{x}),\phi(\mathbf{y})) \\ \mathbf{x}^\top \mathbf{B}\mathbf{y} \end{cases}$$

on k_1, k_2 són funcions kernels en $\mathcal{X} \times \mathcal{X}$ i $\mathcal{X} \in \mathbb{R}^n$, $f(\cdot)$ és una funció real definida en $\mathcal{X}$, la funció $\phi : \mathcal{X} \longrightarrow \mathbb{R}^n$ de transformació, k_3 és un kernel sobre $\mathbb{R}^n \times \mathbb{R}^n$ i per últim $\mathbf{B}$ és una matriu de dimensió $n \times n$ simètrica i definida positiva.

Aquests tipus de operacions les apliquem en els casos en que tenim dos conjunts de dades diferents, d'aquesta forma podrem aplicar a cadascun d'ells una funció kernel diferent i visualitzar-ho tot amb un mateix resultat, combinant els kernels.

3.5 Support Vector Machine (SVM)

En aquest apartat anem a veure un exemple d'un mètode del *machine learning* en el que es poden aplicar les funcions kernel, aquest mètode s'anomena màquines de suport vectorial. Les **màquines de suport vectorial** (*Support Vector Machines*, SVM) són un conjunt d'algorismes d'aprenentatge supervisat, relacionats amb problemes de classificació i regressió. Donat un conjunt d'exemples d'entrenament (de mostres) podem etiquetar les classes i entrenar una SVM per construir un model que predigui la classe d'una nova mostra.

Intuïtivament, una SVM és un model que representa els punts de la mostra en l'espai, separant les classes a 2 espais el més amplis possibles mitjançant un hiperplà de separació. Aquest hiperplà es defineix com el vector entre els 2 punts més pròxims de les dues classes i s'anomena **vector suport**. Quan apliquem noves mostres sobre aquest model, es classifiquen segons l'espai al que pertanyen.

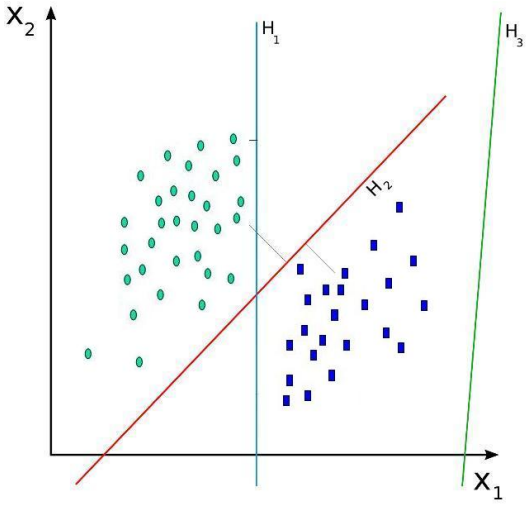

Figura 6: Representació gràfica dels hiperplans separadors del SVM

Idea general

Donat un conjunt de punts $S = (\mathbf{x}_1, \ldots, \mathbf{x}_m) \subset \mathcal{X}$ divisible en dues categories, un algorisme basat en SVM construeix un model capaç de predir a quina categoria pertany nou punt (sense classificació prèvia).

La SVM busca un hiperplà que separi de forma òptima els punts d'una classe i de l'altra, que eventualment han pogut ser prèviament projectats a un espai de dimensió superior. Aquest hiperplà és aquell que té la màxima distància amb els punts que estiguin més pròxims a ell mateix. D'aquesta forma, els punts del conjunt de dades que estan classificats amb una categoria estaran a un costat del hiperplà i els casos que es trobin a l'altre categoria estaran a l'altre costat.

En la figura 6, es pot veure un exemple en dos dimensions de un conjunt de dades agrupades en dos categories (verdes i blaves) separades per dos possibles hiperplans. És clar que existeix un número infinit de possibles hiperplans que realitzin la classificació. La SVM s'encarrega de trobar l'hiperplà òptim, és a dir, aquell que permet un marge màxim entre els elements de les dues categories.

Es denominen vectors de suport als punts que confirmen les dues línies paral·leles al hiperplà, essent la distància entre elles la més gran possible.

3.5.1 Funcions kernel en SVM

La manera més simple de realitzar la separació de les classes en el *support vector machine* és mitjançant una línia recta, un pla o un hiperplà n-dimensional. Desafortunadament els casos a estudiar no acostumen a ser idíl·lics, i el SVM ha de lidiar amb més de dos variables predictores, corbes no lineals de separació, casos on els conjunts de dades no poden ser completament separats o classificadors en més de dos categories. La representació per mitjà de funcions kernel suposa una solució per la majoria d'aquests problemes.

Així doncs, en el SVM amb funcions kernel projectarem l'espai d'entrada en un espai de Hilbert on les tècniques lineals del SVM seran més òptimes.

Avantatges

1. Efectius en espais d'altes dimensions.
2. Segueix essent efectiu en casos en el que la dimensió del espai és més gran que el número de mostres.
3. Utilitza un subconjunt de dades de test en la funció de decisió (anomenats *support vector*), per tant, també és eficient de memòria.
4. És versàtil: es poden utilitzar diferents funcions kernel com a funció de decisió.

Desavantatges

1. Si el número de característiques és molt més gran que el número de mostres, el mètode tendeix a donar resultats pobres.
2. SVM no proveeix de estimacions probabilístiques directament, sinó que aquestes es calculen utilitzant *cross-validation*.

4 Kernel PCA

Com hem vist en els capítols anteriors, hi ha diferents tipus d'algorismes o mètodes pel Machine Learning, els principals són l'aprenentatge supervisat i el no supervisat. L'aprenentatge no supervisat es caracteritza per que els elements de la mostra no contenen etiquetes que els classifiquin. Un dels mètodes que trobem dins d'aquest tipus d'aprenentatge és l'anàlisis de components principals (PCA), és una tècnica de reducció de la dimensió que representen les observacions d'una mostra de grans dimensions en un espai reduït (2 o 3 dimensions) intentant conservar la màxima informació de l'espai original. Anem a aplicar les funcions Kernel en l'aprenentatge no supervisat, en particular en l'anàlisis de components principals. Parlarem del Kernel PCA (KPCA) quan utilitzarem funcions kernel com a dades d'entrada en el mètode d'anàlisi de components principals.

4.1 Anàlisi de components principals (PCA)

L'anàlisi de components principals és un mètode de reducció de la dimensionalitat que busca generar un subespai de menor dimensió que l'espai original. A partir de les n variables originals es creen $r < n$ noves variables, combinacions lineals de les originals, correlacionades entre elles i que conserven la màxima variabilitat de les dades.

Aquestes noves variables són els valors propis i els vectors propis de la matriu de covariàncies de les dades originals centrades. El valor propi més elevat representa la variància de la primera component principal (PC1) i és la combinació lineal de les variables originals amb màxima variància; el vector propi de la matriu de covariàncies són els coeficients de cada variable en la primera component principal.

El següent pas és el càlcul dels coeficients de la segona component principal, aquests són el vector propi de la matriu de covariància amb el segon valor propi més gran i, que a més a més, sigui ortogonal al primer vector propi. D'aquesta forma, totes les components principals estan incorrelacionades.

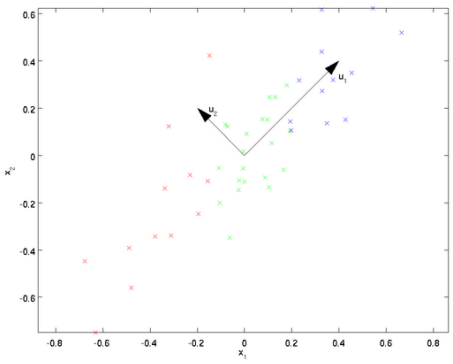

Figura 7: Representació de l'anàlisi de components principals

En la figura 7 podem veure un conjunt de dades en l'espai original i la direcció de les dues primeres components.

Com podem veure, aquests nou eixos de coordenades representen millor la relació entre les variables.

Podem trobar més informació sobre aquest mètode en la bibliografia. A més a més, es pot calcular el PCA automàticament a partir d'algoritmes implementats en R. Trobem vàries llibreries que ens calculen l'anàlisi de components principals, com `stats` (funcions `prcomp` i `princomp`), `FactoMineR` (funció `PCA`), `ade4` (funció `dudi.pca`) i `amap` (funció `acp`).

4.2 Kernel PCA

Per l'anàlisis de components principals amb Kernels apliquem la mateixa idea, però en comptes de fer servir la matriu de covariàncies per detectar el subespai de màxima variabilitat i mínima dimensió, s'utilitza la matriu de kernel $\mathbf{K}$. El que es pretén és trobar un espai de Hilbert de tal forma que les distàncies entre les variables reflecteixin les diferències entre les observacions. Per aconseguir-ho, projectem els punts $\phi(\mathbf{x}_i)$ en un espai lineal de dimensió 2 o 3 de $\mathcal{F}_k$.

Aleshores es pot demostrar que

- és possible determinar els valors propis de la matriu de productes escalars $\mathbf{K}$ centrada i la dispersió de les projeccions dels individus sobre les direccions de màxima variabilitat de l'espai vectorial $\mathcal{F}_k$.
- les projeccions dels individus en aquestes direccions de màxima variabilitat es calculen a partir de $\mathbf{K}$.

Anem a veure aquest mètode amb més detall.

Donat un conjunt de dades $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_m$ amb $\mathbf{x}_i \in \mathcal{X}\ \forall i \in \{1, \ldots, m\}$ i $\mathcal{X} \subseteq \mathbb{R}^n$, es pot escriure aquest conjunt en forma matricial com

$$\mathbf{X} = \begin{pmatrix} x_{11} & \ldots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \ldots & x_{mn} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_m \end{pmatrix}$$

Aleshores $\phi(\mathbf{X}) = \phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_m)$ és la projecció dels elements del conjunt de dades en un nou espai $\mathcal{F}$, on ϕ és la funció *embedding* de la definició 3.1.2. Si $\widetilde{\phi}(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \overline{\phi}$ són les dades centrades en $\mathcal{F}$, la matriu de covariàncies en $\mathcal{F}$ es defineix com:

$$\mathbf{C}_F = \frac{1}{m} \sum_{i=1}^{m} \widetilde{\phi}(\mathbf{x}_i) \widetilde{\phi}(\mathbf{x}_i)^\top$$

Per tant, segons el PCA, hem de trobar els valors propis no nuls d'aquesta matriu. Haurem de resoldre l'equació $\lambda_l \mathbf{v}_l = \mathbf{C}_F \mathbf{v}_l$, on λ_l el l-èssim valor propi de la matriu i $\mathbf{v}_l$ representa el seu respectiu vector propi.

Definim els vectors propis com una combinació lineal de característiques

$$\mathbf{v}_l = \sum_{i=1}^{m} \alpha_l^i \widetilde{\phi}(\mathbf{x}_i) \qquad l = 1, \ldots, r$$

i multiplicant als dos costats per $\widetilde{\phi}(\mathbf{x}_k)$ transposat, obtindrem

$$\lambda_l \sum_{i=1}^{m} \alpha_l^i \langle \widetilde{\phi}(\mathbf{x}_k), \widetilde{\phi}(\mathbf{x}_i) \rangle = \frac{1}{m} \sum_{i=1}^{m} \alpha_l^i \langle \widetilde{\phi}(\mathbf{x}_k), \sum_{j=1}^{m} \widetilde{\phi}(\mathbf{x}_j) \rangle \langle \widetilde{\phi}(\mathbf{x}_j), \widetilde{\phi}(\mathbf{x}_i) \rangle$$

$\forall k \in \{1, \ldots, m\} \qquad l = 1, \ldots, r$

Definim la matriu $\widetilde{\mathbf{K}}$ tal que els seus elements compleixin que $\widetilde{k_{ij}} = \langle \widetilde{\phi}(\mathbf{x}_j), \widetilde{\phi}(\mathbf{x}_i) \rangle$, aleshores obtenim

$$n\lambda_l \widetilde{\mathbf{K}} \alpha_l = \widetilde{\mathbf{K}}^2 \alpha_l \qquad l = 1, \ldots, r$$

Degut a la simetria de $\widetilde{\mathbf{K}}$, els seus vectors propis generen l'espai complet, per tant, $n\lambda_l$ són els valors propis associats a α_l. És per aquest motiu que serà necessari normalitzar els vectors propis de $\widetilde{\mathbf{K}}$:

$$1 = \sum_{i,j=1}^{m} \alpha_l^i \alpha_l^j \langle \widetilde{\phi}(\mathbf{x}_j), \widetilde{\phi}(\mathbf{x}_i) \rangle = \lambda_l \langle \alpha_l, \alpha_l \rangle$$

Les components principals del kernel es podran calcular utilitzant:

$$\langle \mathbf{v}_l, \widetilde{\phi}(\mathbf{x}) \rangle = \sum_{i=1}^{m} \alpha_l^i \langle \widetilde{\phi}(\mathbf{x}_i), \widetilde{\phi}(\mathbf{x}) \rangle$$

per $l = 1, \ldots, r$, on r és el número de vectors propis diferents de zero.

Com ja hem vist en apartats anteriors, normalment ϕ no es coneix, per tant, no es possible calcular la matriu $\widetilde{\mathbf{K}}$, però sí que és possible calcular una matriu equivalent a partir del producte escalar de ϕ, és a dir, a partir de les funcions Kernel. Definim funció kernel general per centrar les dades:

$$\begin{aligned} \widetilde{k}_{ij} &= \langle \widetilde{\phi}(\mathbf{x}_i), \widetilde{\phi}(\mathbf{x}_j) \rangle \\ &= \langle \phi(\mathbf{x}_i) - \overline{\phi}, \phi(\mathbf{x}_j) - \overline{\phi} \rangle \\ &= (\phi(\mathbf{x}_i) - \tfrac{1}{m} \textstyle\sum_{i=1}^{m} \phi(\mathbf{x}_i))(\phi(\mathbf{x}_i) - \tfrac{1}{m} \textstyle\sum_{i=1}^{m} \phi(\mathbf{x}_i))^\top \\ &= k_{ij} - \tfrac{1}{m} \textstyle\sum_{l=1}^{m} k_{il} - \tfrac{1}{m} \textstyle\sum_{l=1}^{m} k_{jl} + \tfrac{1}{m^2} \textstyle\sum_{l,t=1}^{m} k_{lt} \end{aligned} \tag{3}$$

També es pot escriure en forma matricial com:

$$\widetilde{\mathbf{K}} = \mathbf{K} - \frac{1}{m}\mathbf{K}\mathbf{1}_m\mathbf{1}_m^\top - \frac{1}{m}\mathbf{1}_m\mathbf{1}_m^\top\mathbf{K} + \frac{1}{m^2}(\mathbf{1}_m^\top\mathbf{K}\mathbf{1}_m)\mathbf{1}_m\mathbf{1}_m^\top \tag{4}$$

on $\mathbf{1}_m = (1, \ldots, 1)^\top$ és el vector d'uns de dimensió m i $\widetilde{\mathbf{K}}$ és la matriu centrada de $\mathbf{K}$.

Per tant, els vectors escalars $\mathbf{v}_1, \ldots, \mathbf{v}_r$ de $\widetilde{\mathbf{K}}$ són les components principals del kernel PCA:

$$\widetilde{\mathbf{K}}\mathbf{v}_l = \lambda_l\mathbf{v}_l$$

per $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r$. D'aquesta forma no és necessari conèixer ϕ pel càlcul del Kernel PCA, sinó que a partir d'una funció kernel i centrant les dades, podem obtenir les components principals del kernel PCA.

El output final les coordenades cartesianes de $\mathbf{x}_i, \qquad i = 1, \ldots m$ es poden escriure en forma matricial com:

$$\mathbf{A} = \widetilde{\mathbf{K}}\mathbf{V} \tag{5}$$

on $\mathbf{V}$ és la matriu $n \times r$ que conté els vectors propis $\mathbf{v}_l$ de $\widetilde{\mathbf{K}}$.

Observem que es pot representar qualsevol punt addicional $\mathbf{y} \in \mathcal{X}$ en el gràfic de sortida a partir de la projecció de la imatge de $\mathbf{y}$ centrada sobre ϕ en $\mathcal{F}$.

Sigui $\mathbf{Z} = \left(k(\mathbf{y}, \mathbf{x}_i)\right)_{m\times 1}$, aleshores per les equacions [4] i [5] obtenim:

$$\left(\langle\phi(\mathbf{y}) - \overline{\phi}, \mathbf{v}_l\rangle\right) = \left(\mathbf{Z}^\top - \tfrac{1}{m}\mathbf{1}_m^\top\mathbf{K}\right)\left(\mathbf{I}_m - \tfrac{1}{m}\mathbf{1}_m\mathbf{1}_m^\top\right)\mathbf{V}$$

Es pot veure una representació gràfica del Kernel PCA en un conjunt de dades en la imatge 8. A l'esquerra es mostren les dades originals (linealment no separables). Al centre el PCA resultant d'aplicar un kernel gaussià a les dades d'origen, aconseguint una separació lineal de les dades transformades. A la dreta es mostra el resultat d'aplicar un kernel polinòmic, generant una nova distribució espacial de la mostra.

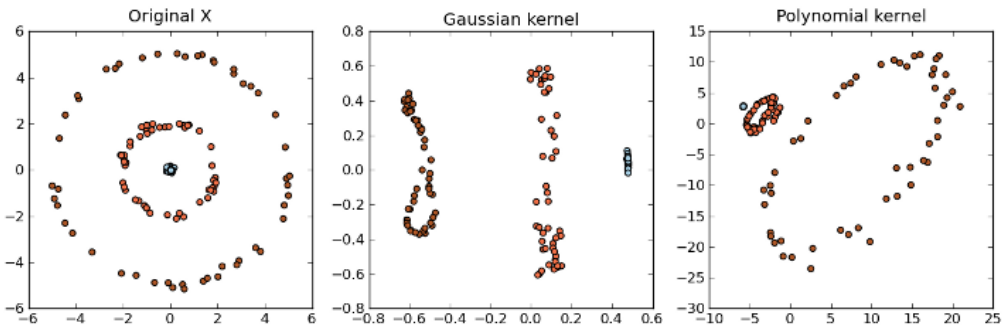

Figura 8: Representació del KPCA per una funció kernel gaussià i una funció kernel polinòmica

Aquest mètode es calcula automàticament amb R a partir de la funció `kpca` de la llibreria `kernlab`, la qual ens proporciona tota la informació relacionada amb l'anàlisi de components principals. A partir d'una matriu kernel, aquesta funció ens proporcionarà la matriu amb les components principals, els valors propis i les coordenades dels punts de les components principals.

Els avantatges de l'ús de funcions kernel en l'anàlisi de components principals és que l'input de les funcions kernel pot ser qualsevol tipus de dades, des de vectors, textos, fins a imatges i el kernel PCA permet l'exploració de relacions no lineals entre les dades.

Tot i els clars avantatges del Kernel PCA per la versatilitat i les propietats associades a les funcions kernel, a la pràctica ens trobem amb alguns inconvenients. Per una banda, la manca d'interpretabilitat de les variables originals, ja que les dades han estat transformades per la funció kernel i estan representades en un nou espai. I per una altra banda, la necessitat de determinar la funció kernel i els paràmetres òptims per a les dades en estudi.

5 Representació de variables originals

Com hem vist en el capítol anterior, el Kernel PCA ofereix una alternativa per a les funcions no lineals, projectant les dades en un espai de alta dimensió. El mètode Kernel ens permet construir diferents versions no lineals de qualsevol algorisme que es pugui expressar en forma de producte escalar. Però com a qualsevol mètode, té els seus desavantatges, el més destacable és la difícil interpretació del output resultant, ja que les variables d'entrada només es gestionen de forma implícita.

En el PCA estàndard, és possible revertir la transformació, recuperant els punts de dades originals i permetent la comprensió de les dades i l'eliminació de soroll. Però en el cas del kernel PCA, com els resultats es troben en un espai de característiques de alta dimensió desconegut, no es possible determinar una representació de les variables en aquest espai.A continuació anem a intentar trobar una solució per aquest inconvenient.

Per millorar la interpretabilitat dels resultat del Kernel PCA, un estudi recentment publicat [5] presenta una nova estratègia de tal forma que és possible representar per cada variable la direcció de màxim creixement a nivell local per cada un dels individus en estudi. Actualment en el departament de Genètica, Microbiologia i Estadística de la Universitat de Barcelona s'està desenvolupant aquest mètode i estan en procés de publicar un nou article, el qual l'autora de la memòria ha tingut el privilegi de poder llegir [4]. En l'article s'exposen els càlculs que venen a continuació.

En l'apartat anterior hem definit un vector $\mathbf{z}$ per poder projectar un punt $\mathbf{y} \in \mathcal{X}$ en l'output del Kernel PCA. Anem a definir ara la corba $\sigma(t) = k(\cdot, \mathbf{x}(t))$, on $\mathbf{x}(t)$ són les solucions de:

$$\frac{dx^\alpha}{dt} = \sum_{\beta=1}^{n} g^{\alpha\beta}(\mathbf{x}) D_\beta f(\mathbf{x}) \qquad \alpha = 1, \ldots, n$$

On $g^{\alpha\beta}(\mathbf{x})$ és la inversa del tensor mètric definit en l'equació [2] i $D_\beta f(\mathbf{x})$ és la primera derivada de $f(\mathbf{x}) \in \mathcal{P}_k$.

Aleshores la projecció de $\mathbf{y} \in \mathcal{X}$ en el Kernel PCA està representada per:

$$\widetilde{\sigma}(t) = \left(\mathbf{Z}^\top - \frac{1}{m}\mathbf{1}_m^\top \mathbf{K}\right)\left(\mathbf{I}_m - \tfrac{1}{m}\mathbf{1}_m\mathbf{1}_m^\top\right)\mathbf{V} \tag{6}$$

on $\mathbf{Z} = \Big(k(\mathbf{y}, \mathbf{x}_i)\Big)_{m\times 1}$. El que ens proposem en aquest apartat és calcular el pendent de $\sigma(t)$ a partir de la seva projecció en $\mathcal{F}$. Això equival a trobar la direcció de màxim creixement d'una variable qualsevol.

És clar que el vector tangent en $t = t_0$ en el cas de que $\mathbf{x}_0 = \phi^{-1}$ o $\sigma(t_0)$ està definit per $\frac{d\sigma}{dt}\|_{t=t_0}$, aleshores la seva projecció en l'espai euclidi en forma matricial és:

$$\left.\frac{d\widetilde{\sigma}}{dt}\right|_{t=t_0} = \left.\frac{d\mathbf{Z}_t^\top}{dt}\right|_{t=t_0}\left(\mathbf{I}_m - \tfrac{1}{m}\mathbf{1}_m\mathbf{1}_m^\top\right)\mathbf{V} \tag{7}$$

Per

$$\left.\frac{d\mathbf{Z}_t^\top}{dt}\right|_{t=t_0} = \left(\left.\frac{d\mathbf{Z}_t^1}{dt}\right|_{t=t_0}, \ldots, \left.\frac{d\mathbf{Z}_t^m}{dt}\right|_{t=t_0}\right)^\top \tag{8}$$

Segons la demostració del manuscrit anteriorment mencionat [4], es té:

$$\begin{aligned}\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} &= \left.\frac{dk(\mathbf{x}(t),\mathbf{x}_i)}{dt}\right|_{t=t_0} \\ &= \sum_{\alpha=1}^{n} D_\alpha k(\mathbf{x}_0,\mathbf{x}_i) \sum_{\beta=1}^{n} g^{\alpha\beta}(\mathbf{x}_0) D_\beta f(\mathbf{x}_0) \\ &= \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n} D_{n+\alpha} k(\mathbf{x_i},\mathbf{x_0}) g^{\alpha\beta}(\mathbf{x_0}) D_\beta f(\mathbf{x_0})\end{aligned} \tag{9}$$

on

1. $D_{n+\alpha}k(\mathbf{x_i},\mathbf{x_0})$ és la derivada de $k(\mathbf{x_i},\mathbf{x_0})$ per la component α de $\mathbf{x}_0$.
2. $g^{\alpha\beta}(\mathbf{x_0})$ és la inversa del tensor mètric definit en l'equació 2. En el cas de que la matriu sigui diagonal es pot escriure com:

$$g^{\alpha\beta}(\mathbf{x}_0) = 2\frac{1}{D_{\alpha\beta}h_{\mathbf{x}_0}(\mathbf{0})}$$

3. $f(\mathbf{x_0})$ és un component de l'espai de vectors $\mathcal{P}_k$, en general considerarem f com una funció lineal i, per tant, serà de la forma $f(\mathbf{x}) = \sum_{\beta=1}^{n} a_\beta x^\beta + c$. En aquest cas, podrem definir

$$D_\alpha f(\mathbf{x}_0) = a_\alpha$$

En conclusió, aquesta derivada ens permet representar les variables en el gràfic del Kernel PCA com a vectors que indiquen la direcció de màxim creixement de cada variable.

Aquesta representació també és aplicable per a combinació lineal de Kernels, de tal forma que representem vectors que indiquen la direcció de màxim creixement per cada combinació lineal.

En els següents capítols podrem veure exemples del càlcul d'aquesta derivada per a diferents funcions kernel, però ara, com a exemple il·lustratiu anem a aplicar aquesta metodologia en un conjunt de dades.

5.1 Exemple: expressió gènica en Ratolins

Per il·lustrar el procediment del mètode del kernel PCA es mostrarà un exemple d'un estudi nutrigenòmic en 40 ratolins [5]. En aquest estudi s'analitza l'efecte de

5 dietes lipídiques diferents en dos grups de ratolins (*wild-type* i mutats (PPARα)). Per fer-ho es recullen fins a 120 gens i 21 àcids grassos. És a dir, el conjunt de dades està compost per dos matrius de dades, en la primera es recull la informació dels 120 gens i en la segona el 21 àcids grassos, les dades estan segmentades per dietes i pel tipus de ratolí (salvatge o mutat).

Les dietes que es recollien es diferenciaven en el tipus d'oli: oli de blat de moro per la dieta de referència (ref), oli de coco (coc), oli de gira-sol (sun), oli de llinosa i oli de peix (fish).

El propòsit de l'estudi és comprovar si canvis en l'expressió gènica impliquen canvis en el àcids grassos o viceversa.

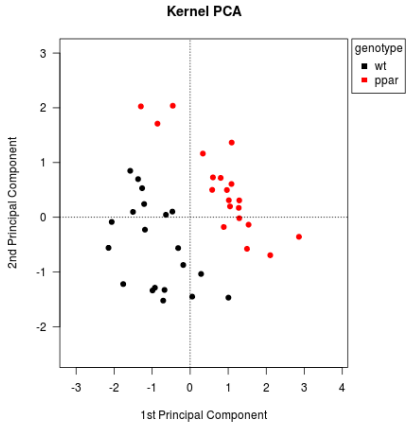

Figura 9: Representació d'un anàlisi de kernel PCA

En primer lloc s'ha representat en la figura 9 el Kernel PCA per als dos genotips, en color vermell podem veure el PPARα i en color negre el WT. Aquests dos gens estan clarament separats. Si només obtinguéssim aquesta informació, es clar, que l'estudi quedaria molt pobre i no es podria continuar amb la interpretació.

En canvi, el output del Kernel PCA amb la representació de variables originals per al kernel Gaussià de la figura 10 podem obtenir una representació més àmplia de les dades. Es poden identificar variables que es comporten de forma similar entre les observacions i, particularment, que apunten cap a una agrupació d'observacions, indicant diferències en els valors d'aquesta variable entre grups. Com es pot veure, es parteix del mateix gràfic del Kernel PCA però s'afegeix la informació de les variables originals. Els vectors representen la direcció de màxim creixement per a dos gens de la mostra (AOX en vermell i CAR1 en verd). La longitud dels vectors varia en funció de la variable que es representi, ja que dependrà de la situació d'aquesta variable en l'espai de projecció. Es pot observar que el gen AOX augmenta cap a *wide-type*, mentre que CAR1 augmenta cap a PPARα.

Per tant, els ratolins que presenten el gen AOX tendeixen a ser del tipus *wild-type*, mentre que els que presenten el gen CAR1 és més probable que siguin del tipus mutant.

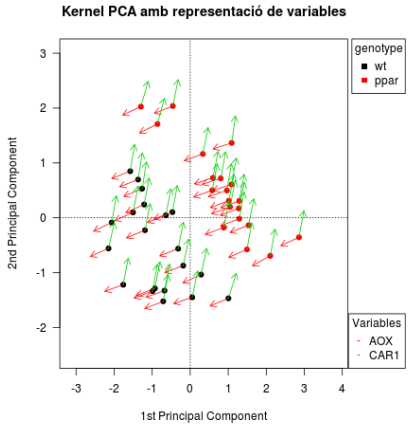

Figura 10: Representació d'un anàlisi de kernel PCA amb la representació de variables originals

5.2 Cerca de variables d'interès

El kernel PCA permet representar un vector per cada observació que mostra la direcció de màxim creixement d'una variable en un punt donat. Es pot pensar, que si definim una direcció d'interès en el pla podem obtenir aquelles variables representades en el kernel PCA que es correlacionin amb aquest vector.

En la figura 11 podem veure aquesta idea representada en les dades dels ratolins de l'exemple anterior. Tornem a veure els dos grups de genotips (vermell i negre), però hem definit un vector $\mathbf{w}$ entre els punts centrals (centroides) de cada clúster. A partir d'aquest vector es vol determinar quines variables estan associades a cada un d'aquests genotips, és a dir, es fa una recerca de les variables que millor es correlacionen amb el vector, essent variables altament implicades en la distribució espacial de les mostres.

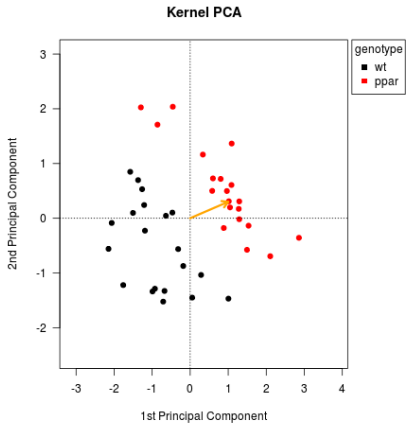

Figura 11: Representació d'un anàlisi kernel PCA amb la cerca de variables d'interès

Per obtenir un valor de correlació per cada variable es calcula la mitjana de les correlacions existents de cada individu entre el vector donat **w** i el vector que representa la direcció de màxim creixement definit anteriorment.

6 Projecció de variables originals segons la funció Kernel

En el següent capítol es mostren els càlculs necessaris per l'obtenció de la projecció de les variables originals per casos particulars de kernel. Les funcions kernel que apareixen en aquest capítol s'han extret de diferents fonts, però la majoria provenen de l'article de Marc G. Genton [2] i de l'article de Yunqian Ma i Guodong Guo [9], on s'agrupen les funcions kernel pel machine learning segons una perspectiva estadística. En aquest treball podrem veure algunes d'aquestes agrupacions i els seus principals exemples de funcions kernel.

A més a més, es calcularà la direcció de màxim creixement de cada variable per diferents exemples de funcions kernel. Per fer-ho s'utilitzaran les equacions del tensor mètric [2], de la distància al quadrat de l'espai de Hilbert [1] i la projecció d'una variable en el kernel PCA [9]. Tots aquests càlculs els farem per la component i de la mostra. Sigui $\mathbf{x}_i$ l'individu i de $\mathcal{X}$ i sigui $\mathbf{x}_0$ un individu qualsevol de la mostra, aleshores aplicarem la funció kernel per $k(\mathbf{x}_i, \mathbf{x}_0)$.

Pel càlcul del tensor mètric definim un vector qualsevol $\mathbf{z} \in \mathbb{R}^n$.

Notem que $\mathbf{x}_i$ pot ser igual a $\mathbf{x}_0$

6.1 Kernel Additiu

Les funcions kernel additives es caracteritzen perquè es poden escriure com la suma de kernels computats per cada dimensió individual de les dades. Una funció general d'un kernel additiu està definida per:

$$K(\mathbf{x}_i, \mathbf{x}_0) = \sum_{\alpha=1}^{n} k(x_i^\alpha, x_0^\alpha)$$

on $k(x^\alpha, y^\alpha)$ és el que s'anomena funció de kernel *dimension-wise*. Aquest tipus de kernels s'utilitzen principalment per classificació d'imatges.

Anem a veure alguns exemples d'aquestes funcions.

6.1.1 Kernel Lineal

La funció Kernel lineal està definida per el producte escalar $\langle , \rangle$ de $\mathbb{R}^n$

$$k(\mathbf{x}_i, \mathbf{x_0}) = \langle \mathbf{x}_i, \mathbf{x_0} \rangle \tag{10}$$

Amb $\mathbf{x}, \mathbf{x_0}$ dos vectors qualsevol de $\mathbb{R}^n$.

Aquest Kernel és la funció més senzilla i equival al mètode de PCA, sense aplicar cap funció kernel. És a dir, la variable original coincideix amb la nova variable al fer la transformació. El Kernel lineal s'utilitza quan es disposa de dades molt diverses.

El seu principal inconvenient és que només es pot aplicar en dades numèriques definides com a vectors.

Anem a calcular la direcció de màxim creixement per la funció de kernel lineal $\mathbf{x}_i \longmapsto k(\mathbf{x}_i, \mathbf{x}_0) \qquad i = 1, \dots m$. En primer lloc veiem la derivada parcial de k respecte la component α del vector $\mathbf{x}_0$.

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = x_i^\alpha$$

El següent pas és calcular el tensor mètric, per això primer definim la funció $h_x(\mathbf{z})$:

$$h_x(\mathbf{z}) = ||\mathbf{x}_0||^2 + ||\mathbf{x}_0 + \mathbf{z}||^2 - 2\langle \mathbf{x}_0, \mathbf{x}_0 + \mathbf{z}\rangle$$

La seva primera derivada respecte la component α de $\mathbf{z}$ és:

$$\frac{\partial h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha} = 2(x_0^\alpha + z^\alpha) - 2x_0^\alpha$$

Aleshores la segona derivada és:

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha \partial z^\beta} = \begin{cases} 0 & \alpha \neq \beta \\ 2 & \alpha = \beta \end{cases}$$

Per tant, segons l'equació 2 el tensor mètric és $g_{\alpha\beta}(\mathbf{x}_0) = \delta_{\alpha\beta}$.

Arribats a aquest punt, ja hem obtingut tots els elements necessaris per trobar la derivada de $\mathbf{Z}$, que és la necessària per representar les variables originals i és de la que farem l'estudi a continuació.

$$\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} = \sum_{\alpha=1}^{n} x_i^\alpha a_\alpha \tag{11}$$

Aquesta derivada és la component i del vector $\frac{dZ}{dt}$, per tant, haurem d'aplicar aquest càlcul per cada $i = 1, \dots, m$

Observació 6.1. Estem suposant que la funció $f(\mathbf{x})$ és lineal.

6.1.2 Kernel Hellinger

El kernel de Hellinger es basa en la distància de Hellinger, definida per la mètrica $\mathcal{D}^2(\mathbf{x}_i, \mathbf{x}_0) = ||\sqrt{\mathbf{x}_i} - \sqrt{\mathbf{x}_0}||_2^2$ per a dos vectors $\mathbf{x}_i, \mathbf{x}_0$ qualsevol. La funció està definida com:

$$k(\mathbf{x}_i, \mathbf{x_0}) = 4\sum_{\alpha=1}^{n} \sqrt{x_i^\alpha x_0^\alpha} \tag{12}$$

Volem trobar la derivada de $\mathbf{Z}$ per poder representar les variables originals, per això necessitem dos elements: el primer és la derivada de la funció de Kernel i el segon és el tensor mètric. Comencem per la derivada de k.

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = \frac{2x_i^\alpha}{\sqrt{x_i^\alpha x_0^\alpha}}$$

Per poder trobar el tensor mètric, primer hem de definir la distància quadrada en l'espai de Hilbert:

$$h_{\mathbf{x}_0}(\mathbf{z}) = 4\sum_{\alpha=1}^{n} x_0^\alpha + 4\sum_{\alpha=1}^{n}(x_0^\alpha + z^\alpha) - 8\sum_{\alpha=1}^{n}\sqrt{x_0^\alpha(x_0^\alpha + z^\alpha)}$$

La seva primera derivada és:

$$\frac{\partial h_{\mathbf{x}_0}}{\partial z^\alpha}(\mathbf{z}) = 0 + 4 + 4\frac{x_0^\alpha}{\sqrt{x_0^\alpha(x_0^\alpha + z^\alpha)}}$$

Estudiem la segona derivada per casos i obtenim el següent resultat:

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha \partial z^\beta} = \begin{cases} 0 & \alpha \neq \beta \\ 2\frac{x_0^{\alpha\,2}}{\sqrt{(x_0^\alpha(x_0^\alpha + z^\alpha))^3}} & \alpha = \beta \end{cases}$$

Per tant, ja podem obtenir el tensor mètric:

$$g_{\alpha\beta}(\mathbf{x_0}) = \frac{1}{x_0^\alpha}\delta_{\alpha\beta}$$

D'aquesta forma hem obtingut tots els elements que necessitàvem i ja podem obtenir la funció

$$\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} = 2\sum_{\alpha=1}^{n}\sqrt{\frac{x_i^{\alpha 3}}{x_0^\alpha}} \tag{13}$$

6.1.3 χ^2-Kernel

El kernel de χ^2 és una elecció molt popular per la formació de SVM no lineal en aplicacions de visió per ordinador. La funció és:

$$k(\mathbf{x}_i, \mathbf{x_0}) = \sum_{\alpha=1}^{n}\frac{2x_i^\alpha x_0^\alpha}{x_i^\alpha + x_0^\alpha} \tag{14}$$

Les dades han de ser no negatives i normalment són normalitzades per tenir la norma de l^1 igual a 1. La normalització es racionalitza amb la connexió a la distància de χ^2, que és una distància entre distribucions de probabilitat discretes.

El χ^2-kernel s'utilitza sobretot en histogrames de paraules visuals.

La derivada d'aquesta funció per la component α pels elements $\mathbf{x}_i, \mathbf{x}_0$ de $\mathcal{X}$ és

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = \frac{2x_i^\alpha}{(x_0^\alpha + x_i^\alpha)^2}$$

A continuació fem el càlcul del tensor mètric. Definim la distància quadrada del espai de Hilbert

$$h_{\mathbf{x}_0}(\mathbf{z}) = \sum_{\alpha=1}^{n} x_0^\alpha + \sum_{\alpha=1}^{n}(x_0^\alpha + z^\alpha) - 2\sum_{\alpha=1}^{n}\frac{2x_0^\alpha(x_0^\alpha + z^\alpha)}{2x_0^\alpha + z^\alpha}$$

Fem el càlcul de la seva derivada per la component α

$$\frac{\partial h_{\mathbf{x}_0}}{\partial z^\alpha}(\mathbf{z}) = 1 - 4\frac{x_0^{\alpha 2}}{(2x_0^\alpha + z^\alpha)^2}$$

Separem la segona derivada per casos. Suposem que $\alpha = \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^{\alpha 2}}(\mathbf{z}) = 8\frac{x_0^{\alpha 2}}{(2x_0^\alpha + z^\alpha)^3}$$

Suposem que $\alpha \neq \beta$.

$$\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^\alpha \partial z^\beta}(\mathbf{z}) = 0$$

Per tant, quan $\mathbf{z} = \mathbf{0}$ obtenim el tensor mètric d'aquest Kernel, que és $g_{\alpha\beta}(\mathbf{x_0}) = \frac{1}{2}\frac{1}{x_0^\alpha}\delta_{\alpha\beta}$. D'aquesta forma obtenim tots els elements necessaris per calcular la matriu de derivades.

$$\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} = 2\sum_{\alpha=1}^{n}\frac{2x_i^\alpha x_0^\alpha}{(x_i^\alpha + x_0^\alpha)^2}a_\alpha \tag{15}$$

6.1.4 Generalized histogram Intersection kernel

Histogram intersection kernel s'han introduït recentment en les tasques de reconeixement d'imatges, segons l'article publicat per Sabri Boughorbel [13]. La funció la podem trobar a continuació:

$$k(\mathbf{x}_i, \mathbf{x_0}) = \sum_{\alpha=1}^{n}\min(|x_i^\alpha|^p, |x_0^\alpha|^q) \tag{16}$$

Per a $p, q \in \mathbb{N}$ qualsevol. Aquest kernel és definit positiu per qualsevol valor de p, q.

Una de les principals característiques d'aquest kernel en l'anàlisi de reconeixement d'imatges és que les imatges a analitzar poden ser de qualsevol mida. A més a més, el límit d'aquest kernel és invariant segons la dimensió del *feature space* pel SVM.

La derivada d'aquesta funció és

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = \begin{cases} q|x_0^\alpha|^{q-1} & |x_0^\alpha|^q < |x_i^\alpha|^p \\ 0 & altrament \end{cases}$$

Anem a fer el càlcul del tensor mètric. Primer definim la funció

$$h_{\mathbf{x}_0}(\mathbf{z}) = \sum_{i=1}^n \min(|x_0^i|^p, |x_0^i|^q) + \sum_{i=1}^n \min(|x_0^i + z^i|^p, |x_0^i + z^i|^q) - 2\sum_{i=1}^n \min(|x_0^i|^p, |x_0^i + z^i|^q)$$

Notem que pels dos primers sumatoris, la única diferència entre els dos valors del mínim és el valor de p i q. Per tant, podem suposar que $p < q$ (el resultat serà equivalent per $p > q$). En aquest cas la funció té el següent resultat:

$$h_{\mathbf{x}_0}(\mathbf{z}) = \sum_{i=1}^n |x_0^i|^p + \sum_{i=1}^n |x_0^i + z^i|^p - 2\sum_{i=1}^n \min(|x_0^i|^p, |x_0^i + z^i|^q)$$

La seva derivada en funció a l'element α és:

$$\frac{\partial h_{\mathbf{x}_0}}{\partial z^\alpha}(\mathbf{z}) = \begin{cases} p(x_0^\alpha + z^\alpha)^{p-1} - 2q(x_0^\alpha + z^\alpha)^{q-1} & |x_0^\alpha|^p > |x_0^\alpha + z^\alpha|^q \\ 0 & altrament \end{cases}$$

Suposarem a partir d'aquest moment que la primera derivada no és nul·la. Per calcular la segona derivada d'aquesta funció anem a separar per casos. Suposem primer que $\alpha = \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^{\alpha 2}}(\mathbf{z}) = p(p-1(x_0^\alpha + z^\alpha)^{p-2} - 2q(q-1)(x_0^\alpha + z^\alpha)^{q-2}$$

Sigui $\alpha \neq \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^\alpha \partial z^\beta}(\mathbf{z}) = 0$$

Quan $\mathbf{z} = \mathbf{0}$, tenim dues opcions:

$$g_{\alpha\beta}(\mathbf{x}_0) = \begin{cases} \frac{1}{2}(p(p-1)(x_0^\alpha)^{p-2} - 2q(q-1)(x_0^\alpha)^{q-2}) & |x_0^\alpha|^q < |x_0^\alpha|^p \\ \frac{1}{2}p(p-1)(x_0^\alpha)^{p-2} & |x_0^\alpha|^q > |x_0^\alpha|^p \end{cases}$$

Com que hem suposat que $p < q$, si $\mathbf{z} = \mathbf{0}$ mai es complirà que $|x_0^\alpha|^q < |x_0^\alpha|^p$ per $p, q \in \mathbb{N}$ i, per tant, el tensor mètric és: $g_{\alpha\beta}(\mathbf{x}_0) = \frac{1}{2}p(p-1)(x_0^\alpha)^{p-2}\delta_{\alpha\beta}$

Per últim:

$$\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} = \frac{2}{p-1}\sum_{\alpha=1}^n a_\alpha x_0^\alpha \tag{17}$$

6.2 Kernel Estacionari

Les funcions kernel estacionàries són aquelles que només depenen de la diferència entre dos vectors $\mathbf{x}_i, \mathbf{x}_0$ qualsevol i no d'aquests per si mateixos. Les funcions són de la forma:

$$k(\mathbf{x}_i, \mathbf{x}_0) = k_s(\mathbf{x}_i - \mathbf{x}_0)$$

L'ús d'aquest tipus de kernel s'ha estès en els darrers anys, sobretot en les sèries temporals i la estadística espacial, perquè permeten fer una inferència en k per a tot parell de punts separats pel mateix vector.

Quan un kernel estacionari només depèn de la norma entre entre dos punts $\mathbf{x}_i, \mathbf{x}_0$ i no en la direcció d'aquests, aleshores s'anomena **kernel isotròpic**:

$$k(\mathbf{x}_i, \mathbf{x}_0) = k_I(||\mathbf{x}_i - \mathbf{x}_0||)$$

Anem a veure alguns exemples de funcions kernel isotròpiques.

6.2.1 Kernel Gaussià

L'equació del Kernel Gaussià és:

$$k(\mathbf{x}_i, \mathbf{x_0}) = \exp\Big(-\frac{||\mathbf{x}_i - \mathbf{x_0}||^2}{\theta^2}\Big) \tag{18}$$

on $||\cdot||$ és la norma euclídea.

El paràmetre θ permet controlar la flexibilitat d'aquest Kernel.

- $\theta \to 0$ la funció assignarà valors molt propers a zero a les parelles formades pels elements de l'espai d'entrada, encara que les diferències siguin molt petites. En aquest cas la matriu Kernel s'assemblarà molt a la matriu identitat i tindrem problemes d'*overfitting*.
- $\theta \to \infty$ la funció assignarà valors molt pròxims a 1, de manera quer ens trobarem davant d'una funció pràcticament constant i no serà útil per la detecció de patrons.

La flexibilitat d'aquest Kernel fa que sigui un dels més usats, sobretot quan no es té informació prèvia de les dades.

Anem a calcula la direcció de màxim creixement de les variables per aquest Kernel. Com sempre, el primer pas és obtenir la seva derivada.

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = \frac{2(x_i^\alpha - x_0^\alpha)}{\theta^2} k(\mathbf{x_i}, \mathbf{x_0})$$

El següent pas és definir la funció $h_x(\mathbf{z})$ i trobar la seva segona derivada respecte les components (z^α, z^β).

$$h_{\mathbf{x}_0}(\mathbf{z}) = 2 - 2\exp\Big(-\frac{||\mathbf{z}||^2}{\theta^2}\Big)$$

La derivada és:

$$\frac{\partial h_{\mathbf{x}_0}}{\partial z^\alpha}(\mathbf{z}) = 4\frac{z^\alpha}{\theta^2}k(\mathbf{z},\mathbf{0})$$

Hem de separar la segona derivada segons els valors de α i β. Suposem que $\alpha \neq \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha \partial z^\beta} = \frac{-8z^\alpha z^\beta}{\theta^4}k(\mathbf{z},\mathbf{0})$$

Suposem que $\alpha = \beta$.

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha \partial z^\beta} = \frac{4}{\theta^2}k(\mathbf{z},\mathbf{0}) - \frac{8z^{\alpha 2}}{\theta^4}k(\mathbf{z},\mathbf{0})$$

Podem veure que en el cas de que $\alpha \neq \beta$, per

$$\mathbf{z} = \mathbf{0}$$

la segona derivada ens dóna 0, pero en el cas de que $\alpha = \beta$, obtenim $\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^\alpha \partial z^\beta}(\mathbf{0}) = \frac{4}{\theta^2}$

Aleshores només podem escollir el cas de que $\alpha = \beta$ i el tensor mètric d'aquest Kernel és $g_{\alpha\beta}(\mathbf{x_0}) = \frac{2}{\theta^2}\delta_{\alpha\beta}$. D'aquesta forma obtenim tots els elements necessaris per calcular la matriu de derivades.

$$\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} = k(\mathbf{x_i},\mathbf{x_0})\sum_{\alpha=1}^{n}(x_i^\alpha - x_0^\alpha)a_\alpha \tag{19}$$

6.2.2 Rational Quadratic Kernel

El kernel racional quadràtic té el mateix caràcter de la combinació lineal de funcions kernel gaussianes. La majoria de persones que estudien un procés Gaussià de regressió o classificació acaben fent serivr la funció Gaussiana o el kernel racional quadràtic. Aquesta és una sol·lució ràpida que funciona molt bé per interpolar funcions suaus i contínues. És a dir, quan l'algorisme en que apliquem la funció kernel està basat en una funció suau o contínua. La funció kernel racional quadràtic presenta alguns problemes quan les dades són més grans que bidimensionals, ja que fa difícil detectar errors en el model.

El kernel racional quadràtic es diferència del kernel Gaussià en que és menys costós computacionalment i pot ser una bona alternativa quan el kernel Gaussià es torna molt car. La seva funció és:

$$k(\mathbf{x}_i,\mathbf{x_0}) = 1 - \frac{||\mathbf{x}_i - \mathbf{x_0}||^2}{||\mathbf{x}_i - \mathbf{x_0}||^2 + \theta} \tag{20}$$

Suposem que $\theta \neq 0$ i que $||\mathbf{x}_i - \mathbf{x_0}|| < \theta \ \forall \mathbf{x}_i, \mathbf{x}_0 \in \mathcal{X}$. El primer pas és el càlcul de la derivada de $k(\mathbf{x}_i, \mathbf{x}_0)$ respecte l'element α de $\mathbf{x}_0$.

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = \frac{2\theta(x_i^\alpha - x_0^\alpha)}{(||\mathbf{x}_i - \mathbf{x}_0||^2 + \theta)^2}$$

El següent pas és definir la funció

$$h_{\mathbf{x}_0}(\mathbf{z}) = 2\frac{||\mathbf{z}||^2}{||\mathbf{z}||^2 + \theta}$$

i trobar les seves derivades.

$$\frac{\partial h_{\mathbf{x}_0}}{\partial z^\alpha}(\mathbf{z}) = \frac{4\theta z^\alpha}{||\mathbf{z}||^2 + \theta)^2}$$

Separem la segona derivada per casos. Suposem primer que $\alpha = \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^{\alpha 2}} = 4\theta\frac{||\mathbf{z}||^2 + \theta - 4z^{\alpha 2}}{(||\mathbf{z}||^2 + \theta)^3}$$

Suposem ara que $\alpha \neq \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha \partial z^\beta} = -16\theta\frac{z^\alpha z^\beta}{(||\mathbf{z}||^2 + \theta)^3}$$

Quan $\mathbf{z} = \mathbf{0}$, la segona derivada per $\alpha \neq \beta$ és zero, per tant, $g_{\alpha\beta}(\mathbf{x_0}) = \frac{2}{\theta}\delta_{\alpha\beta}$. D'aquesta forma obtenim el resultat que volíem

$$\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} = \frac{\theta^2}{(||\mathbf{x}_i - \mathbf{x}_0||^2 + \theta)^2}\sum_{\alpha=1}^{n}(x_i^\alpha - x_0^\alpha)a_\alpha \tag{21}$$

6.2.3 Kernel de Wave

Aquest Kernel és simètric i semidefinit positiu. La seva propietat més destacable és que és una funció simètrica.

$$k(\mathbf{x}_i, \mathbf{x_0}) = \frac{\theta}{||\mathbf{x}_i - \mathbf{x_0}||}\sin\left(-\frac{||\mathbf{x}_i - \mathbf{x_0}||}{\theta}\right) \tag{22}$$

Suposem que $\theta \neq 0$ i que $||\mathbf{x}_i - \mathbf{x_0}|| < \theta \ \forall \mathbf{x}_i, \mathbf{x}_0 \in \mathcal{X}, i = 1, \ldots, n$. A continuació es pot veure una representació d'aquest kernel.

La derivada de la funció del Kernel de Wave amb respecte la component α pels elements $\mathbf{x}_i, \mathbf{x}_0$ de $\mathcal{X}$ és la que es mostra a continuació.

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = \frac{\theta(x_i^\alpha - x_0^\alpha)}{||\mathbf{x}_i - \mathbf{x}_0||^3}\sin\left(-\frac{||\mathbf{x}_i - \mathbf{x}_0||}{\theta}\right) + \frac{(x_i^\alpha - x_0^\alpha)}{||\mathbf{x}_i - \mathbf{x}_0||^2}\cos\left(-\frac{||\mathbf{x}_i - \mathbf{x}_0||}{\theta}\right)$$

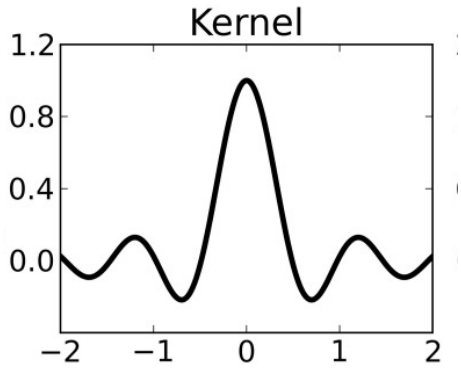

Figura 12: Imatge de la funció del kernel de Wave

A continuació fem el càlcul del tensor mètric. Definim la funció

$$h_{\mathbf{x}_0}(\mathbf{z}) = -2\frac{\theta}{||\mathbf{z}||}\sin\Big(-\frac{||\mathbf{z}||}{\theta}\Big)$$

Per trobar la direcció de màxim creixement, resoldrem les derivades de la distància al quadrat de Hilbert $h(\mathbf{z})$ per Taylor. Aleshores aquesta funció es pot definir com:

$$h_{\mathbf{x}_0}(\mathbf{z}) = -2\sum_{j=0}^{\infty}(-1)^j\frac{||\mathbf{z}||^{2j}}{\theta^{2j}(2j+1)!} = -2\Big(1-\frac{||\mathbf{z}||^2}{\theta^2 3!}+\frac{||\mathbf{z}||^4}{\theta^4 5!}-\dots\Big)$$

Fem el càlcul de la seva derivada per la component α del vector $\mathbf{z}$

$$\frac{\partial h_{\mathbf{x}_0}}{\partial z^\alpha}(\mathbf{z}) = 4\frac{z^\alpha}{\theta^2 3!} - 8\frac{z^\alpha||\mathbf{z}||^2}{\theta^4 5!}+\dots$$

Separem la segona derivada per casos. Suposem que $\alpha = \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^{\alpha 2}}(\mathbf{z}) = \frac{4}{\theta^2 3!} - 8\frac{||\mathbf{z}||^2}{\theta^4 5!} - 16\frac{z^{\alpha 2}}{\theta^4 5!}+\dots$$

Suposem que $\alpha \neq \beta$.

$$\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^\alpha \partial z^\beta}(\mathbf{z}) = -16\frac{z^\alpha z^\beta}{\theta^4 5!}$$

Com podem veure, per als dos casos, quan la $\mathbf{z} = \mathbf{0}$ i $\alpha = \beta$, aleshores la segona derivada és zero, però quan $\alpha = \beta$ obtenim que el tensor mètric és: $g_{\alpha\beta} = \frac{1}{3\theta^2}\delta_{\alpha\beta}$.

Observació 6.2. Notem que no és necessari el càlcul de la derivada per la resta del sumatori, ja que quan $\mathbf{z} = \mathbf{0}$ el resultat sempre serà zero.

Per tant, la direcció de màxim creixement per la component $i = 1, \dots, m$ i l'individu $\mathbf{x}_0$ és:

$$\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} = \Big(\frac{3\theta^3}{||\mathbf{x}_i - \mathbf{x}_0||^3}\sin\Big(-\frac{||\mathbf{x}_i - \mathbf{x}_0||}{\theta}\Big) - \frac{3\theta^2}{||\mathbf{x}_i - \mathbf{x}_0||^2}\cos\Big(-\frac{||\mathbf{x}_i - \mathbf{x}_0||}{\theta}\Big)\Big)\sum_{\alpha=1}^{n}(x_i^\alpha - x_0^\alpha)a_\alpha \tag{23}$$

6.2.4 Multiquadratic Kernel

Aquest kernel es pot fer servir pels mateixos casos que el kernel racional quadràtic i és un exemple d'un kernel definit no positiu.

$$k(\mathbf{x}_i, \mathbf{x_0}) = \sqrt{||\mathbf{x}_i - \mathbf{x_0}||^2 + \theta^2} \tag{24}$$

Suposem que $\theta \neq 0$ i que $||\mathbf{x}_i - \mathbf{x_0}|| < \theta \ \forall \mathbf{x}_i, \mathbf{x}_0 \in \mathcal{X}$. Calculem la primera derivada d'aquesta funció respecte la component α de un dels elements de la base de dades $\mathbf{x}_0, \mathbf{x}_i \in \mathcal{X}$

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = -\frac{x_i^\alpha - x_0^\alpha}{\sqrt{||\mathbf{x}_i - \mathbf{x}_0||^2 + \theta^2}}$$

El següent pas és trobar el tensor mètric. Per això primer s'ha de definir la funció $h_{\mathbf{x}_0}(\mathbf{z})$ i calcular les seves derivades.

$$h_{\mathbf{x}_0}(\mathbf{z}) = 2\theta - 2\sqrt{||\mathbf{z}||^2 + \theta^2}$$

La primera derivada d'aquesta funció respecte la component α de z és:

$$\frac{\partial h_{\mathbf{x}_0}}{\partial z^\alpha}(\mathbf{z}) = -2\frac{z^\alpha}{\sqrt{||\mathbf{z}||^2 + \theta^2}}$$

Separem per casos. Suposem primer que $\alpha = \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^{\alpha 2}} = -2\frac{||\mathbf{z}||^2 + \theta^2 - z^{\alpha 2}}{\sqrt{(||\mathbf{z}||^2 + \theta^2)^3}}$$

Suposem ara que $\alpha \neq \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha \partial z^\beta} = 2\frac{z^\alpha z^\beta}{\sqrt{(||\mathbf{z}||^2 + \theta^2)^3}}$$

Com podem veure, si $\mathbf{z} = \mathbf{0}$, la segona derivada per $\alpha \neq \beta$ és zero, així doncs podem escriure el tensor mètric com $g_{\alpha\beta}(\mathbf{x_0}) = \frac{1}{\theta}\delta_{\alpha\beta}$. Així doncs, podem calcular el que estàvem buscant.

$$\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} = \frac{\theta}{\sqrt{||\mathbf{x}_i - \mathbf{x}_0||^2 + \theta}} \sum_{\alpha=1}^{n} (x_i^\alpha - x_0^\alpha) a_\alpha \tag{25}$$

6.2.5 Inverse Multiquadratic Kernel

Aquest kernel té una propietat interessant i és que tots els seus vectors propis són linealment independents. A més a més, la dimensió del *feature space* pot ser il·limitada i, igual que amb el kernel Gaussià, la matriu de kernel té rang màxim. La funció està definida per:

$$k(\mathbf{x}_i, \mathbf{x_0}) = \frac{1}{\sqrt{||\mathbf{x}_i - \mathbf{x_0}||^2 + \theta^2}} \tag{26}$$

Suposem que $\theta \neq 0$ i que $||\mathbf{x}_i - \mathbf{x_0}|| < \theta \ \forall \mathbf{x}_i, \mathbf{x}_0 \in \mathcal{X}$. Comencem pel càlcul de la derivada del Kernel.

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = \frac{x_i^\alpha - x_0^\alpha}{\sqrt{(||\mathbf{x}_i - \mathbf{x}_0||^2 + \theta^2)^3}}$$

El següent pas és trobar el tensor mètric, per fer-ho calculem la segona derivada de la funció

$$h_{\mathbf{x}_0}(\mathbf{z}) = \frac{2}{\theta} - 2\frac{1}{\sqrt{||\mathbf{z}||^2 + \theta^2}}$$

La primera derivada d'aquesta funció és

$$\frac{\partial h_{\mathbf{x}_0}}{\partial z^\alpha}(\mathbf{z}) = 2\frac{z^\alpha}{\sqrt{(||\mathbf{z}||^2 + \theta^2)^3}}$$

Separem per casos. Suposem primer que $\alpha = \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^{\alpha 2}} = 2\frac{||\mathbf{z}||^2 + \theta^2 - 3z^{\alpha 2}}{\sqrt{(||\mathbf{z}||^2 + \theta^2)^5}}$$

Suposem ara que $\alpha \neq \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha \partial z^\beta} = -6\frac{z^\alpha z^\beta}{\sqrt{(||\mathbf{z}||^2 + \theta^2)^5}}$$

Com podem veure, si $\mathbf{z} = \mathbf{0}$, la segona derivada per $\alpha \neq \beta$ és zero, així doncs podem escriure el tensor mètric com $g_{\alpha\beta}(\mathbf{x_0}) = \frac{1}{\theta^3}\delta_{\alpha\beta}$. Així doncs, podem calcular el que estàvem buscant.

$$\frac{dZ_t^i}{dt}\Big|_{t=t_0} = \frac{\theta^3}{(\sqrt{||\mathbf{x}_i - \mathbf{x}_0||^2 + \theta})^3} \sum_{\alpha=1}^{n} (x_i^\alpha - x_0^\alpha) a_\alpha \tag{27}$$

6.2.6 Poisson Kernel

Encara que no és el típic kernel estacionari, s'ha decidit afegir en aquesta secció per que només depèn de les diferències entre els punts i, per tant, entra dins de la definició de kernel estacionari. La distribució de Poisson sorgeix com la distribució de una variable aleatòria registrant el número de ocurrències de un esdeveniment estrany amb una taxa mitjana constant, per un període o població donat. La funció de kernel de Poisson (28) està basat en aquesta distribució.

$$k(\mathbf{x}_i, \mathbf{x_0}) = 4 \exp\Big(-\frac{1}{2} ||\sqrt{\mathbf{x}} - \sqrt{\mathbf{x_0}}||^2 \Big) \tag{28}$$

La derivada d'aquesta funció per a dues components cualsevols $\mathbf{x}_i, \mathbf{x}_0$ és:

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = \frac{1}{2} k(\mathbf{x_i}, \mathbf{x_0}) \Big(\frac{\sqrt{x_i^\alpha}}{\sqrt{x_0^\alpha}} - 1 \Big)$$

A continuació fem el càlcul del tensor mètric. Definim la funció

$$h_{\mathbf{x}_0}(\mathbf{z}) = -8 \exp\Big(-\frac{1}{2} ||\sqrt{\mathbf{x}_0} - \sqrt{\mathbf{x}_0 + z}||^2 \Big)$$

Fem el càlcul de la seva derivada per la component α

$$\frac{\partial h_{\mathbf{x}_0}}{\partial z^\alpha}(\mathbf{z}) = -4 \exp\Big(-\frac{1}{2} ||\sqrt{\mathbf{x}_0} - \sqrt{\mathbf{x}_0 + \mathbf{z}}||^2 \Big) \Big(\sqrt{\frac{x_0^\alpha}{x_0^\alpha + z^\alpha}} - 1 \Big)$$

Separem la segona derivada per casos. Suposem que $\alpha = \beta$

$$\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^{\alpha 2}}(\mathbf{z}) = 2 \exp\Big(-\frac{1}{2} ||\sqrt{\mathbf{x}_0} - \sqrt{\mathbf{x}_0 + \mathbf{z}}||^2 \Big) \Big(\sqrt{\frac{x_0^\alpha}{(x_0^\alpha + z^\alpha)^3}} - \Big(\sqrt{\frac{x_0^\alpha}{x_0^\alpha + z^\alpha}} - 1 \Big)^2 \Big)$$

Suposem que $\alpha \neq \beta$.

$$\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^\alpha \partial z^\beta}(\mathbf{z}) = -2 \exp\Big(-\frac{1}{2} ||\sqrt{\mathbf{x}_0} - \sqrt{\mathbf{x}_0 + \mathbf{z}}||^2 \Big) \Big(\sqrt{\frac{x_0^\alpha}{x_0^\alpha + z^\alpha}} - 1 \Big) \Big(\sqrt{\frac{x_0^\beta}{x_0^\beta + z^\beta}} - 1 \Big)$$

Quan $\mathbf{z} = \mathbf{0}$ tenim que $g_{\alpha\beta}(\mathbf{x_0}) = x_0^\alpha \delta_{\alpha\beta}$ és el tensor mètric. Aleshores la direcció de màxim creixement de la variable és:

$$\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} = \frac{1}{2}k(\mathbf{x_i}, \mathbf{x_0}) \sum_{\alpha=1}^{n} \left(\frac{\sqrt{x_i^\alpha}}{\sqrt{x_0^\alpha}} - 1\right) x_0^\alpha a_\alpha \tag{29}$$

6.3 Kernel no Estacionari

La classe més general de funcions kernel són els no estacionari, els quals depenen explícitament dels dos punts $\mathbf{x}, \mathbf{x}_0$:

$$k(\mathbf{x}_i, \mathbf{x_0})$$

La majoria d'aquestes funcions contenen el producte escalar entre els dos punts $< \mathbf{x}, \mathbf{x}_0 >$

Anem a veure alguns exemples sobre kernel no estacionari.

6.3.1 Kernel Polinòmic

La funció polinòmica computa un polinomi de grau $d \in \mathbb{N}$ entre dos vectors. Aquesta funció representa la similitud entre dos vectors. Conceptualment, el kernel polinòmic no només té en compte la similitud entre dos vectors sota la mateixa dimensió, sinó que també a través de dimensions. Es fa servir en algorismes de *machine learning* ja que fa un comptatge de la interacció entre característiques.

La funció Kernel polinòmica està definida per $c, \theta \in \mathbb{R}$ constants positives reals i $d \in \mathbb{N}$ constant no nul·la.

$$k(\mathbf{x}_i, \mathbf{x_0}) = (\theta \langle \mathbf{x}_i, \mathbf{x_0} \rangle + c)^d \tag{30}$$

Notem que quan $d = 1$, $\theta = 1$ i $c = 0$ es redueix a un Kernel lineal.

Primer comencem amb la derivada parcial d'aquesta funció kernel respecte la component α del vector $\mathbf{x}_0$.

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = d\theta x_i^\alpha (\theta \langle \mathbf{x}_i, \mathbf{x_0} \rangle + c)^{d-1}$$

Hem d'intentar definir la funció $g_{ij}(x)$, per això definim primer la següent funció per a un $\mathbf{z}$ vector qualsevol de $\mathbb{R}^n$:

$$h_{\mathbf{x}_0}(\mathbf{z}) = (\theta ||\mathbf{x}_0||^2 + c)^d + (\theta ||\mathbf{x}_0 + \mathbf{z}||^2 + c)^d - 2(\theta \langle \mathbf{x}_0, \mathbf{x}_0 + \mathbf{z} \rangle + c)^d$$

D'aquesta forma obtenim que

$$\frac{\partial h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha} = 2d\theta(x_0^\alpha + z^\alpha)(\theta ||\mathbf{x}_0 + \mathbf{z}||^2 + c)^{d-1} - 2d\theta x_0^\alpha (\theta \langle \mathbf{x}_0, \mathbf{x}_0 + \mathbf{z} \rangle + c)^{d-1}$$

Separem per casos per la segona derivada. Suposem que $\alpha \neq \beta$, aleshores el resultat que obtenim és:

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha \partial z^\beta} = 4d(d-1)\theta^2(x_0^\alpha + z^\alpha)(x_0^\beta + z^\beta)d(d-1)\theta^2 x_0^\alpha x_0^\beta(\theta\langle \mathbf{x}_0, \mathbf{x}_0 + \mathbf{z}\rangle + c)^{d-2}$$

En el cas de que $\alpha = \beta$, obtenim:

$$\begin{aligned}\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^{\alpha 2}} &= d\theta(\theta||\mathbf{x}_0 + \mathbf{z}||^2 + c)^{d-1} \\ &+ 4d(d-1)\theta^2(x_0^\alpha + z^\alpha)^2(\theta||\mathbf{x}_0 + \mathbf{z}||^2 + c)^{d-2} \\ &- 2d(d-1)\theta^2 x_0^{\alpha 2}(\theta\langle \mathbf{x}_0, \mathbf{x}_0 + \mathbf{z}\rangle + c)^{d-2}\end{aligned}$$

Per tant,

$$g_{\alpha\beta}(\mathbf{x_0}) = d(d-1)\theta^2 x_0^\alpha x_0^\beta(\theta||\mathbf{x}_0||^2 + c)^{d-2} + d\theta(\theta||\mathbf{x}_0||^2 + c)^{d-1}\delta_{\alpha\beta}$$

Un cop ja tenim tots els càlculs necessaris, ho apliquem tot a la equació 9 i d'aquesta forma obtenim:

$$\begin{aligned}\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} &= \frac{(\theta\langle \mathbf{x}_i, \mathbf{x_0}\rangle + c)^{d-1}}{(\theta||\mathbf{x}_0||^2 + c)^{d-2}} \sum_{\alpha=1}^{n} \sum_{\substack{\beta=1 \\ \beta\neq\alpha}}^{n} \frac{x_i^\alpha}{(d-1)\theta x_0^\alpha x_0^\beta} a_\beta \\ &+ \frac{(\theta\langle \mathbf{x}_i, \mathbf{x_0}\rangle + c)^{d-1}}{(\theta||\mathbf{x}_0||^2 + c)^{d-2}} \sum_{\alpha=1}^{n} \frac{x_i^\alpha}{(d-1)x_0^{\alpha 2} + \theta||\mathbf{x}_0||^2 + c} a_\alpha\end{aligned} \tag{31}$$

6.3.2 Kernel Anova

El kernel Anova va ser definit per Vapnik i Wahba. La funció Kernel Anova està definida per una constant $d \in \mathbb{N}_{>0}$:

$$k(\mathbf{x}_i, \mathbf{x_0}) = \Big(\sum_{\alpha=1}^{n} \exp\Big(-\theta(x_i^\alpha - x_0^\alpha)^2\Big)\Big)^d \tag{32}$$

La constant d indica l'ordre d'interaccions entre els atributs x_i^α i és millor escollir un nombre petit. Aquest Kernel ha demostrat funcionar molt bé per la resolució de problemes de regressió multidimensional.

El primer pas per trobar la seva representació pel Kernel PCA és fer el càlcul de la seva derivada parcial respecte la component x_0^α

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = d2\theta(x_i^\alpha - x_0^\alpha)\exp(-\theta(x_i^\alpha - x_0^\alpha)^2)\Big(\sum_{\alpha=1}^{n} \exp(-\theta(x_i^\alpha - x_0^\alpha)^2)\Big)^{d-1}$$

El següent pas és calcular el tensor mètric. Per l'equació (2) sabem que primer hem de trobar la segona derivada en funció de la component α-èssima i β-èssima de un vector $\mathbf{z}$ de $\mathbb{R}^n$ de la funció

$$h_{\mathbf{x}_0}(\mathbf{z}) = n^d + n^d - 2\Big(\sum_{\alpha=1}^{n} \exp\Big(-\theta z^{\alpha 2}\Big)\Big)^d$$

Fent els càlculs necessaris obtenim:

$$\frac{\partial h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha} = 4d\theta z^\alpha \exp\Big(-\theta z^{\alpha 2}\Big)\Big(\sum_{\alpha=1}^{n} \exp\Big(-\theta z^{\alpha 2}\Big)\Big)^{d-1}$$

Separem per casos, suposem que $\alpha = \beta$. Anomenem $\Delta = \sum_{\alpha=1}^{n} \exp(-\theta z^{\alpha 2})$. Aleshores la segona derivada és:

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^{\alpha 2}} = 4d\theta\Delta^{d-2} \exp\Big(-\theta z^{i 2}\Big)(2(d-1)\theta \exp\Big(-\theta z^{\alpha 2}\Big) z^{\alpha 2} + 2\theta z^{\alpha 2}\Delta - \Delta)$$

En el cas de que $\alpha \neq \beta$ la derivada és:

$$\frac{\partial^2 h_{\mathbf{x}_0}(\mathbf{z})}{\partial z^\alpha \partial z^\beta} = 8d(d-1)\theta^2 z^\alpha z^\beta \Delta^{d-2} \exp\Big(-\theta z^{\alpha 2}\Big) \exp\Big(-\theta z^{\beta 2}\Big)$$

En el cas de que $\mathbf{z} = \mathbf{0}$, obtenim que $\Delta = n$ i a més a més, si $\alpha \neq \beta$ la segona derivada és nul·la. D'aquesta forma obtenim el tensor mètric:

$$g_{\alpha\beta}(\mathbf{x_0}) = 2\theta d n^{d-1}\delta_{\alpha\beta}$$

on $\delta_{\alpha\beta}$ és la funció delta de Kronecker. Per tant, ja podem trobar l'equació que ens interessa.

$$\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} = \frac{1}{n^{d-1}} \exp\Big(-\theta(\mathbf{x}_i - \mathbf{x}_0)^2\Big)\Big(\sum_{i=1}^{n} \exp\Big(-\theta(\mathbf{x}_i - \mathbf{x}_0)^2\Big)\Big)^{d-1} \sum_{\alpha=1}^{n} (x_i^\alpha - x_0^\alpha) a_\alpha \tag{33}$$

6.3.3 Hyperbolic tangent (Sigmoid) Kernel

El kernel de Sigmoid és un model de SVM molt semblant a una xarxa neuronal amb dues capes de perceptrons. La funció és la següent:

$$k(\mathbf{x}_i, \mathbf{x_0}) = \tanh(\theta\langle \mathbf{x}_i, \mathbf{x_0}\rangle + c) \tag{34}$$

La derivada d'aquesta funció és:

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = \frac{\theta x_i^\alpha}{\cosh^2(\theta\langle \mathbf{x}_i, \mathbf{x_0}\rangle + c)}$$

Anem a calcular el tensor mètric d'aquesta funció:

$$h_{\mathbf{x}_0}(\mathbf{z}) = \tanh(\theta||\mathbf{x}_0||^2 + c) + \tanh(\theta||\mathbf{x}_0 + \mathbf{z}||^2 + c) - 2\tanh(\theta\langle \mathbf{x}_0, \mathbf{x}_0 + \mathbf{z}\rangle + c)$$

La primera derivada és:

$$\frac{\partial h_{\mathbf{x}_0}}{\partial z^\alpha}(\mathbf{z}) = 2\theta\frac{(x_0^\alpha + z^\alpha)}{\cosh^2(\theta||\mathbf{x}_0 + \mathbf{z}||^2 + c)} - \frac{2\theta x_0^\alpha}{\cosh^2(\theta\langle \mathbf{x}_0, \mathbf{x}_0 + \mathbf{z}\rangle + c)}$$

Per calcular la segona derivada separem per casos. Suposem primer que $\alpha = \beta$

$$\begin{aligned}\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^{\alpha 2}}(\mathbf{z}) = &\frac{\theta}{\cosh^2(\theta||\mathbf{x}_0 + \mathbf{z}||^2 + c)} - 8\theta^2(x_0^\alpha + z^\alpha)^2\frac{\tanh(\theta||\mathbf{x}_0 + \mathbf{z}||^2 + c)}{\cosh^2(\theta||\mathbf{x}_0 + \mathbf{z}||^2 + c)} \\ &+ 4\theta^2 x_0^{\alpha 2}\frac{\tanh(\theta\langle \mathbf{x}_0, \mathbf{x}_0 + \mathbf{z}\rangle + c)}{\cosh^2(\theta\langle \mathbf{x}_0, \mathbf{x}_0 + \mathbf{z}\rangle + c)}\end{aligned}$$

Suposem ara que $\alpha \neq \beta$

$$\begin{aligned}\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^\alpha \partial z^\beta}(\mathbf{z}) = &-8\theta^2(x_0^\alpha + z^\alpha)(x_0^\beta + z^\beta)\frac{\tanh(\theta||\mathbf{x}_0 + \mathbf{z}||^2 + c)}{\cosh^2(\theta||\mathbf{x}_0 + \mathbf{z}||^2 + c)} \\ &+ 4\theta^2 x_0^\alpha x_0^\beta\frac{\tanh(\theta\langle \mathbf{x}_0, \mathbf{x}_0 + \mathbf{z}\rangle + c)}{\cosh^2(\theta\langle \mathbf{x}_0, \mathbf{x}_0 + \mathbf{z}\rangle + c)}\end{aligned}$$

Per tant, el tensor mètric es divideix en dos casos:

$$g_{\alpha\beta}(\mathbf{x}_0) = \begin{cases} \frac{1}{\cosh^2(\theta||\mathbf{x}_0||^2 + c)} - 2\theta^2(x_0^\alpha)^2\frac{\tanh(\theta||\mathbf{x}_0||^2 + c)}{\cosh^2(\theta||\mathbf{x}_0||^2 + c)} & \alpha = \beta \\ -2\theta^2 x_0^\alpha x_0^\beta\frac{\tanh(\theta||\mathbf{x}_0||^2 + c)}{\cosh^2(\theta||\mathbf{x}_0||^2 + c)} & \alpha \neq \beta \end{cases}$$

En aquest cas ja podem donar la direcció de màxim creixement de les variables per aquest Kernel:

$$\begin{aligned}\left.\frac{dZ_t^i}{dt}\right|_{t=t_0} = &\frac{-\cosh^2(\theta||\mathbf{x}_0||^2 + c)}{2\theta\cosh^2(\theta\langle \mathbf{x}_i, \mathbf{x_0}\rangle + c)\tanh(\theta||\mathbf{x}_0||^2 + c)}\sum_{\alpha=1}^n\sum_{\substack{\beta=1 \\ \beta\neq\alpha}}^n\frac{x_i^\alpha}{x_0^\alpha x_0^\beta}a_\alpha \\ &+ \frac{\cosh^2(\theta||\mathbf{x}_0||^2 + c)}{\cosh^2(\theta\langle \mathbf{x}_i, \mathbf{x_0}\rangle + c)}\sum_{\alpha=1}^n\frac{\theta x_i^\alpha}{1 - 2\theta^2 x_0^{\alpha 2}\tanh(\theta||\mathbf{x}_0||^2 + c)}\end{aligned} \tag{35}$$

6.4 Wavelet Kernel

Aquest kernel és diferent a la resta de funcions kernel vistes fins al moment perquè està definit a partir d'una funció $h(x)$. El Wavelet kernel (Zhang et al, 2004) prové de la teoria de Wavelet i es defineix com:

$$k(\mathbf{x}_i, \mathbf{x_0}) = \prod_{j=1}^{n} h\Big(\frac{x_i^j - c}{a}\Big) h\Big(\frac{x_0^j - c}{a}\Big) \tag{36}$$

Siguin a, b els coeficients de dilatació i translació de Wavelet respectivament i $h(x)$ la *mother function* de Wavelet.

La funció kernel Wavelet ens és útil per la millora de l'escassetat d'aproximació, permetent-nos construir una funció de kernel adequada per les nostres dades. Normalment, aquesta funció s'utilitza conjuntament amb el *support vector Machine* per combinar les seves propietats. La funció de kernel Wavelet extrau les característiques del model i el SVM les classifica.

En l'article publicat per Li Zhang, Weide Zu i Lichang Jiao [7], els autors suggereixen com una possible *mother function*:

$$h(x) = \cos(1,75x)\exp(-\frac{x^2}{2})$$

En aquest article, els autors proven que la funció compleix les propietats de les funcions kernel.

Per al cas en que a nosaltres ens interessa,escollirem una funció $h(x)$ qualsevol i calcularem la direcció de màxim creixement per aquesta. La derivada de la funció de Wavelet és:

$$\frac{\partial k}{\partial x_0^\alpha}(\mathbf{x}_i, \mathbf{x}_0) = h'\Big(\frac{x_0^\alpha - c}{a}\Big) h\Big(\frac{x_i^\alpha - c}{a}\Big) \prod_{\substack{j=1 \\ j\neq\alpha}}^{n} h\Big(\frac{x_i^j - c}{a}\Big) h\Big(\frac{x_0^j - c}{a}\Big)$$

Anem a trobar el tensor mètric. Per això definim la distància al quadrat del espai de Hilbert de l'equació 1

$$h_{\mathbf{x}_0}(\mathbf{z}) = \prod_{j=1}^{n} h\Big(\frac{x_0^j - c}{a}\Big)^2 + \prod_{j=1}^{n} h\Big(\frac{x_0^j + z^j - c}{a}\Big)^2 - 2\prod_{j=1}^{n} h\Big(\frac{x_0^j - c}{a}\Big) h\Big(\frac{x_0^j + z^j - c}{a}\Big)$$

La derivada d'aquesta funció respecte la component α de la variable z és:

$$\frac{\partial h_{\mathbf{x}_0}}{\partial z^\alpha}(\mathbf{z}) = 2h'\Big(\frac{x_0^\alpha + z^\alpha - c}{a}\Big)h\Big(\frac{x_0^\alpha + z^\alpha - c}{a}\Big)\prod_{\substack{j=1\\ j\neq\alpha}}^{n} h\Big(\frac{x_0^j + z^j - c}{a}\Big)^2$$
$$- h'\Big(\frac{x_0^\alpha + z^\alpha - c}{a}\Big)h\Big(\frac{x_0^j - c}{a}\Big)\prod_{\substack{j=1\\ j\neq\alpha}}^{n} h\Big(\frac{x_0^j - c}{a}\Big)h\Big(\frac{x_0^j + z^j - c}{a}\Big)$$

Anem a calcular ara la segona derivada d'aquesta funció. Considerem dos casos. Suposem primer que $\alpha = \beta$:

$$\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^{\alpha 2}}(\mathbf{z}) = 2\Big(h''\Big(\frac{x_0^\alpha + z^\alpha - c}{a}\Big)h\Big(\frac{x_0^\alpha + z^\alpha - c}{a}\Big) + h'\Big(\frac{x_0^\alpha + z^\alpha - c}{a}\Big)^2\Big)\prod_{\substack{j=1\\ j\neq\alpha}}^{n} h\Big(\frac{x_0^j + z^j - c}{a}\Big)^2$$
$$- 2\Big(h''\Big(\frac{x_0^\alpha + z^\alpha - c}{a}\Big)h\Big(\frac{x_0^j - c}{a}\Big) + h'\Big(\frac{x_0^\alpha + z^\alpha - c}{a}\Big)^2\Big)\prod_{\substack{j=1\\ j\neq\alpha}}^{n} h\Big(\frac{x_0^j - c}{a}\Big)h\Big(\frac{x_0^j + z^j - c}{a}\Big)$$

Suposem que $\alpha \neq \beta$.

$$\frac{\partial^2 h_{\mathbf{x}_0}}{\partial z^\alpha \partial z^\beta}(\mathbf{z}) =$$
$$4h'\Big(\frac{x_0^\alpha + z^\alpha - c}{a}\Big)h\Big(\frac{x_0^\alpha + z^\alpha - c}{a}\Big)h'\Big(\frac{x_0^\beta + z^\beta - c}{a}\Big)h\Big(\frac{x_0^\beta + z^\beta - c}{a}\Big)\prod_{\substack{j=1\\ j\neq\alpha,\beta}}^{n} h\Big(\frac{x_0^j + z^j - c}{a}\Big)^2$$
$$- 2h'\Big(\frac{x_0^\alpha + z^\alpha - c}{a}\Big)h\Big(\frac{x_0^\beta - c}{a}\Big)h'\Big(\frac{x_0^\beta + z^\beta - c}{a}\Big)h\Big(\frac{x_0^\beta - c}{a}\Big)\prod_{\substack{j=1\\ j\neq\alpha,\beta}}^{n} h\Big(\frac{x_0^j - c}{a}\Big)h\Big(\frac{x_0^j + z^j - c}{a}\Big)$$

Per tant, quan $\mathbf{z} = \mathbf{0}$, en el cas de que $\alpha = \beta$ la segona derivada dóna 0, però en el cas de que $\alpha \neq \beta$ la segona derivada de h dóna un número diferent de 0. Aleshores el tensor mètric és:

$$g_{\alpha\beta}(\mathbf{x_0}) = \begin{cases} 2h'\Big(\frac{x_0^\alpha - c}{a}\Big)h\Big(\frac{x_0^\beta - c}{a}\Big)h'\Big(\frac{x_0^\beta - c}{a}\Big)h\Big(\frac{x_0^\beta - c}{a}\Big)\prod_{\substack{j=1\\ j\neq\alpha,\beta}}^{n} h\Big(\frac{x_0^j - c}{a}\Big)h\Big(\frac{x_0^j - c}{a}\Big) & \alpha \neq \beta \\ 0 & \alpha = \beta \end{cases}$$

Aleshores la direcció de màxim creixement és:

$$\frac{dZ_t^i}{dt}\Big|_{t=t_0} = \frac{1}{2}\prod_{\substack{j=1\\ j\neq\alpha,\beta}}^{n} \frac{h\Big(\frac{x_i^j - c}{a}\Big)}{h\Big(\frac{x_0^j - c}{a}\Big)}\sum_{\beta=1}^{n} \frac{h\Big(\frac{x_i^\beta - c}{a}\Big)}{h'\Big(\frac{x_0^\beta - c}{a}\Big)}a_\beta \tag{37}$$

7 Software Estadístic

El R és un llenguatge orientat al processament i anàlisi de dades. Es caracteritza per ser desenvolupat de forma lliure i gratuïta. El projecte va començar fa més de vint anys i hi participen membres de varis països. Actualment, ofereix un total de 6657 llibreries proporcionant als usuaris diverses eines d'alta qualitat per a l'anàlisi estadístic i la representació gràfica de dades.

Al 2015, amb la intenció d'implementar la representació de variables originals al llenguatge R, la Núria Planell crea la llibreria `KPCAplus` en el seu treball de màster [10]. L'objectiu de la Núria va ser divulgar la representació de les variables originals per millorar la interpretabilitat del kernel PCA i proporcionar eines per a la integració de dades basades en kernels per a les funcions de kernel polinòmic, gaussià i Anova. Aquesta llibreria s'encarrega de:

- Integrar conjunts de dades mitjançant kernels.
- Representar les variables originals en el kernel PCA.
- Cercar variables d'interès.

En aquest treball s'incorporen noves funcions descrites en el capítol anterior per aportar més possibilitats de càlcul a la llibreria `KPCAplus`. A partir d'aquest moment es pot calcular el kernel PCA i la representació de les variables originals per a les funcions kernel que es mostren a la taula. El codi d'aquest paquet es pot veure en l'Annex. Per afegiment, en aquesta taula podem veure el nom necessari a introduir en la funció `KPCAplus` per aplicar-se aquest kernel i els paràmetres de cada funció.

Funció kernel	Nom en `KPCAplus`	Paràmetres
Lineal	`vanilladot`	
Polinic	`polydot`	`scale, offset, degree`
Hellinger	`hellinger`	
Racional Quadràtic	`ratquad`	`offset`
Multiquadràtic	`multquad`	`offset`
Inverse multiquadràtic	`invmultquad`	`offset`
Gaussià	`rbfdot`	`scale`
Anova	`anovadot`	`scale`
Wave	`wave`	`scale`
χ^2	`chiquad`	
Poisson	`poisson`	
Tangent hiperbòlic	`tanhdot`	`scale, offset`

L'objectiu d'ampliar el paquet és poder arribar a una varietat de dades més gran i poder aproximar-nos al millor mètode segons les dades que estudiem, a partir de l'ús de la funció kernel adient.

A continuació es descriuen les funcions principals d'aquest paquet:

7.1 Funció KPCAplus

Aquesta és la funció principal del paquet i és la que relaciona la resta de funcions. S'encarrega de recollir les dades, fer les comprovacions prèvies, generar el kernel PCA i mostra el gràfic d'aquest amb la projecció de les variables originals o bé la cerca de variables d'interès.

```
KPCAplus = function(x, kernel_function, kparameters, xpc=1, ypc=2,
factors=NULL, variables=NULL, combination="+",  var_combination=NULL,
coef_combination=NULL, scale=0.5, plot.kPCA=TRUE, plot.profile=FALSE,
variables_discovery=NULL, discovery.col="orange", weights=NULL,
text.sample=FALSE, show.legend=TRUE, ...)
```

Els principals arguments, necessaris per l'execució de la matriu i el càlcul del KPCA són:

`x`

Matriu de dades indexada per files. Pel cas de voler el KPCA per més d'un conjunt de dades, es donarà una llista de matrius.

`kernel_function`

Funció kernel a utilitzar. El nom de cada funció s'indica en la taula anterior (columna "Nom en `KPCAplus`"). En el cas d'estudiar més d'un conjunt de dades, es donarà una llista amb el kernel de cada conjunt de dades amb el mateix ordre que la llista de matrius de dades `x`. Per tant, el primer kernel s'aplicarà a la primera matriu de dades de `x`.

`kparameters`

Llista amb els paràmetres corresponents al kernel definit a `kernel_function`. Els paràmetres a definir en cada funció kernel es troben en la taula anterior. Pel cas de més d'un conjunt de dades, es donarà una llista de llistes de paràmetres kernel, amb el mateix ordre que `kernel_function`.

`variables`

Nom de les variables dels conjunts de dades analitzades que es volen representar en el el KPCA. Per defecte és `NULL`. En el cas de no donar aquesta informació, es farà el kernel PCA per a totes les variables del conjunt.

`variables_discovery`

Vector numèric de la forma $\mathbf{v} = (x_0, y_0, x_1, y_1)$, indicant les coordenades corresponents a una direcció d'interès en el pla. Per defecte és `NULL`.

L'explicació de la resta d'arguments que trobem en la funció es pot localitzar en l'ajuda del paquet en R. Serveixen per modificar la gràfica del kernel PCA, com el color de les variables, escollir quines components principals es volen representar, determinar el titul del gràfic, etc.

Els valors que obtenim amb aquesta funció és una llista amb:

`datasets`: Recull la informació de les dades analitzades, incloent la matriu `x` i el nombre de conjunts.

`KPCA`: conté tota la informació relacionada amb el kernel PCA: la funció kernel, els paràmetres, la matriu kernel, els valors propis, les components principals i les projeccions del kernel PCA.

`Variable dicovery`: en cas de generar la cerca de variables d'interés, es donarà un `data.frame` de 4 columnes i n files (variables) on, per cada variable, s'arreplegrà el valor mitjà de les correlacions i la desviació estàndard.

7.2 Altres

A continuació es descriuen algunes de les funcions que trobem en el paquet i a les quals crida la funció `KPCAplus` pel càlcul de la projecció de variables originals.

1. `derivation`: aquesta funció calcula la derivada i el tensor mètric per a totes les funcions kernels descrites en la taula anterior. S'ha intentat programar la funció de forma modular per poder ampliar fàcilment amb noves funcions kernel. Retorna el producte de la derivada pel tensor mètric en una llista amb $i \leq n$ matrius de dimensió $m \times m$, on i és el número de variables que es volen estudiar.

2. `derTensor`: calcula el producte de la derivada pel tensor en els casos en que el tensor depèn de $\alpha, \beta = 1, \ldots n$. El que fa es agrupar les matrius de la derivada i del tensor per individus, en comptes de per variables i fer el producte d'aquestes.

3. `gderivation`: calcula la direcció de màxim creixement per a cada conjunt de dades. Retorna una llista amb aquesta direcció per les variables demanades en `KPCAplus` i per cada conjunt de dades.

4. `kernelMatrix_computation`: computa les matrius de kernel per cada conjunt de dades. Per fer-ho crida a la funció `kernelMatrix` externa del paquet `kernlab`. Per les funcions kernel que no es troben en aquest paquet s'han creat noves funcions amb el càlcul de les matrius de kernel. S'ha programat així perque en el cas d'ampliar el paquet amb més funcions kernel sigui més senzill.

8 Conclusions

L'anàlisi de components principals (PCA) és una tècnica clàssica de reducció de la dimensió per casos on les combinacions lineals entre les variables són vàlids. El problema d'aquest mètode és que únicament permet trobar afinitats lineals entre les dades i, actualment, moltes de les relacions existents són no lineals. El *machine learning* és una metodologia molt utilitzada en l'actualitat i en continu desenvolupament, que permet l'estudi de gran quantitat de dades. En particular, va permetre ampliar el mètode PCA a partir dels mètodes basats en funcions kernel, desenvolupant el mètode de kernel PCA, que aplica les funcions kernel en l'anàlisi de components principals i permet trobar relacions no lineals entre les dades.

Per desgràcia, el mètode de kernel PCA encara que molt útil, té una representació gràfica molt pobre, ja que amb aquest es perd tota la informació de les dades originals. És per aquest motiu que en el departament de Genètica, Microbiologia i Estadística de la Universitat de Barcelona es va desenvolupar la projecció de les variables originals en el gràfic del kernel PCA, permetent mostrar la direcció de màxim creixement per cada variable.

Un cop definida la teoria, faltava implementar aquest recurs en algun llenguatge de programació que permetés aplicar-ho a dades reals. A causa d'aquesta necessitat la Núria Planell va crear el paquet en R `KPCAplus`, el qual permetia el càlcul del kernel PCA i la recerca de variables per a un o més conjunts de dades.

Però aquest paquet es va fer insuficient al voler aplicar diferents tipologies de dades, ja que només representava el kernel polinòmic, el kernel gaussià i el kernel Anova. Per aquest fet s'han enumerat diferents funcions kernel i les seves principals propietats i s'ha ampliat el paquet `KPCAplus` amb els càlculs necessaris per aplicar la projecció de variables originals en el kernel PCA per a les funcions descrites.

Amb tot això podem concloure que:

- Els mètodes basats en funcions kernel, per les seves propietats, poden ser una bona aproximació per a la integració de dades, permetent la identificació de relacions no lineals i l'ús de diferents tipus de dades (textos, imatges, números, etc.).
- La metodologia del departament d'Estadística de la Universitat de Barcelona permet millorar la interpretabilitat de les variables en el kernel PCA per un conjunt de dades o per a dades integrades [4].
- La funció desenvolupada en R, `KPCAplus` ofereix la possibilitat de representar el kernel PCA, la projecció de les variables originals i la projecció de variables originals per a diferents funcions kernel sense restricció en el nombre de conjunts de dades a estudiar.
- La funció `KPCAplus` s'ha implementat de forma modular per a poder ser ampliada fàcilment amb més funcions kernel.

- Amb les diferents funcions kernel implementades en R és possible arribar a dades més variades, estenent l'abast de l'anàlisi de components principals en funcions kernel i, per sobre de tot, l'aplicació de la projecció de variables originals.

Tot i la feina realitzada, encara queda molt camí per recórrer dins del món del kernel PCA, la integració de dades i la visualització i cerca de variables d'interès. Ens queda pendent investigar els diferents mètodes per a la determinació de les funcions kernel i els paràmetres del kernel a utilitzar. Addicionalment, manca definir la representació de variables per a un ventall més ampli de funcions kernel, com per exemple per a cadenes de caràcters (string kernel).

Per últim, seria interessant ampliar la llibreria `Shiny` que va començar la Núria Planell en el seu treball de màster i que permetia l'ús d'aquest mètode per a tot tipus d'usuaris, des de novells fins a experts, per a les noves funcions kernel definides en aquest treball.

Referències

[1] Alpaydin, E. (2014). *Introduction to machine learning*. MIT press.

[2] Genton, M. G. (2001). *Classes of kernels for machine learning: a statistics perspective. Journal of machine learning research*, 2(Dec), 299-312.

[3] Venables, W. N., Smith, D. M., R Development Core Team. (2004). *An introduction to R*.

[4] Ferran Reverter, Esteban Vegas, Marta Cubedo, Antonio Miñarro, Miquel Calvo, Gloria García, Martín Ríos i Josep M.Oller (2017). Enhanced plotting in Data Analysis. *Manuscript, unpublished*.

[5] Reverter, F., Vegas, E., Oller, J. M. (2014). Kernel-PCA data integration with enhanced interpretability. *BMC systems biology*, 8(2), 1.

[6] Souza, C. R. (2010). *Kernel functions for machine learning applications*. Creative Commons Attribution-Noncommercial-Share Alike, 3.

[7] Tong, Y., Yang, D., Zhang, Q. (2006). Wavelet kernel support vector machines for sparse approximation. *Journal of Electronics (China)*, 23(4), 539-542.

[8] Hofmann, T., Schölkopf, B., Smola, A. J. (2008). Kernel methods in machine learning. *The annals of statistics*, 1171-1220.

[9] Ma, Y. (2014). *Support vector machines applications*. G. Guo (Ed.). New York: Springer.

[10] Núria Plantell. (2015) Kernel PCA per a l'anàlisi de dades òmiques. Barcelona: *Publicacions i Edicions Universitat de Barcelona*

[11] Boughorbel, S., Tarel, J. P., Fleuret, F., Boujemaa, N. (2005, September). The GCS kernel for SVM-based image recognition. *In International Conference on Artificial Neural Networks* (pp. 595-600). Springer Berlin Heidelberg.

[12] David Duvenaud *The kernel Cookbook: Advice on Covarianve functions*

[13] Boughorbel, S., Tarel, J. P., Boujemaa, N. (2005, September). Generalized histogram intersection kernel for image recognition. *In IEEE International Conference on Image Processing* 2005 (Vol. 3, pp. III-161). IEEE.

[14] Vert, J. P., Tsuda, K., Schölkopf, B. (2004). A primer on kernel methods. *Kernel Methods in Computational Biology*, 35-70.

[15] Mitchell, T. M. (2006). *The discipline of machine learning* (Vol. 9). Carnegie Mellon University, School of Computer Science, Machine Learning Department.

[16] Williams, C. (2006). *Machine Learning and Statistics: What's the Connection?* Belfast, UK.

[17] Wang, Q. (2012). Kernel principal component analysis and its applications in face recognition and active shape models. *arXiv preprint arXiv:1207.3538*.

9 Annex

Codi R de la funció `KPCAplus`

```
#require packages
require(kernlab)
require(gplots)
require(mixOmics)
require(ggplot2)


#------------------------------------------->>>> KPCAplus
   internal functions

derivation=function(x, vars, kernel_function, kparameters,
   kernelM){
#Comput the direction of maximum growth (derivation at s=0)
   of determined variables for each sample
#Input:
#       x: matrix indexed by rows or list of matrixs with row
    data
#       vars: variable names
#       kernel_function: kernel function
#       kparameters: list with kernel parameters
#       kernelM: kernel matrix
#Output: list with the direction of maximum growth (
   derivation at s=0) of determined variables for each sample

cl <- list()
k<-1
for(i in 1:length(vars)){ #for each variable
#Gaussian
if(identical("rbfdot",kernel_function)){ #if gaussian kernel
parms <- unlist(kparameters)
#Derivative
Adif <- t(sapply(x[,vars[i]], FUN=function(aa){aa - x[,vars[i
   ]]}))
der <- (2/parms)*Adif*kernelM #return the derivation for
   gaussian kernel for each data point
#Metric Tensor
g_aa <- 4/parms^2
#Derivative*tensor
res <- der/g_aa
}

if(identical("anovadot",kernel_function)){ #if ANOVA kernel
parms <- unlist(kparameters)
Adif <- t(sapply(x[,vars[i]], FUN=function(aa){x[,vars[i]]-aa
   }))
```

```
e <- exp(-parms["sigma"]*Adif^2)
s <- apply(e, 2, FUN = sum)^(parms["degree"]-1)
der <- 2*parms["degree"]*parms["sigma"]*Adif*e*s #return the
   derivation for anova kernel for each data point
#Metric Tensor
g_aa <- 2*parms["sigma"]*parms["degree"]*length(vars)^(parms[
   "degree"]-1)
#Tensor*Derivative
res <- der/g_aa
}

if(identical("vanilladot", kernel_function)){ #if lineal
   kernel
res <- t(sapply(x[,vars[i]], FUN=function(aa){x[,vars[i]]}))
} #return the derivation for lineal kernel for each data
   point

if(identical("ratquad", kernel_function)){ #if Rational
   Quadratic kernel
parms <- unlist(kparameters)
#Derivative
Adif <- t(sapply(x[,vars[i]], FUN = function(aa){aa-x[,vars[i
   ]]}))
der <- (2*parms["sigma"]*Adif)/(dist(x)^2+parms["sigma"])^2
#Metric Tensor
g_aa <- 2/parms["sigma"]
#Derivative*Metric Tensor
res <- der/g_aa
}

if(identical("multicuadratic", kernel_function)){ #if
   Multiquadratic kernel
#Derivative
parms <- unlist(kparameters)
Adif <- t(sapply(x[,vars[i]], FUN = function(aa){aa-x[,vars[i
   ]]}))
der <- -Adif/sqrt(dist(x)^2+parms["sigma"]^2)
#Metric Tensor
g_aa <- 1/parms["sigma"]
#Derivative*Metric Tensor
res <- der/g_aa
}

if(identical("inverseMult", kernel_function)){ #if inverse
   Multiquadratic kernel
#Derivative
parms <- unlist(kparameters["sigma"])
Adif <- t(sapply(x[,vars[i]], FUN = function(aa){aa-x[,vars[i
   ]]}))
```

```
der <- Adif/sqrt(as.matrix(dist(x, diag=TRUE, upper=TRUE)^2)+
   parms^2)^3
#Metric Tensor
g_aa <- 1/parms["sigma"]^3
#Derivative*Metric Tensor
res <- der/g_aa
}

if(identical("spherical", kernel_function)){ #if Spherical
   kernel
parms <- unlist(kparameters)
#Derivative
Adif <- t(sapply(x[,vars[i]], FUN = function(aa){aa-x[,vars[i
   ]]}))
der <- 3/(2*parms)*Adif*(1/dist(x, diag=TRUE, upper=TRUE)+
   dist(x, diag=TRUE, upper=TRUE)/parms^2)
#Metric Tensor
g_aa <- 3/(2*parms)
#Derivative*tensor
res <- der/g_aa
}

if(identical("hellinger", kernel_function)){#if Hellinger
   kernel
#Derivative
der <- t(sapply(x[,vars[i]], FUN = function(aa){2*aa/sqrt(aa*
   x[,vars[i]])}))
#tensor m tric
g_aa <- 1/x[,vars[i]]
#Derivative * tensor
res <- der/g_aa
}#return the derivation for hellinger kernel for each data
   points


if(identical("chi", kernel_function)){ #if Chi square kernel
#Derivative
der <- t(sapply(x[,vars[i]], FUN = function(aa){2*aa/(aa+x[,
   vars[i]])^2}))
#Metric Tensor
g_aa <- 1/(2*x[,vars[i]])
#Derivative*Metric Tensor
res <- der/g_aa
}#return the derivation for Chi^2 kernel for variable i and
   all x_0.

if(identical("poisson", kernel_function)){ #if Poisson kernel
#Derivative
Asqrt <- t(sapply(x[,vars[i]], FUN = function(aa){sqrt(aa)/
```

```
   sqrt(x[,vars[i]])-1}))
der <- Asqrt/2 * kernelM
#Metric Tensor
g_aa <- x[,vars[i]]
#producte
res <- der/g_aa
}#return the derivation for Poisson kernel.

if(identical("polydot",kernel_function)){ #if polinomial
   kernel
parms <- unlist(kparameters)
g_ab <- list()
scaled_x <- parms["scale"]*x
offscaled_x <- apply(x,1,FUN=function(obs){scaled_x%*%obs}) #
   producto escalar
res <- parms["degree"]*(offscaled_x+parms["offset"])^(parms["
   degree"]-1)*x[,vars[i]] #return the derivation for
   polinomial kernel for each data point
#Metric Tensor
off_g <- apply(x,1,FUN=function(obs){x%*%obs})
x_ab <- apply(x, 2, FUN = function(aa){aa*x})
x_aa <- parms["scale"]*diag(off_g)+parms["offset"]
g_ab <- parms["degree"]*x_aa^(parms["degree"]-2)*(parms["
   degree"]-1)*parms["scale"]*x_ab
for(n in 1:nrow(x)){
g_ab[k,i] <- g_ab[k,i] + parms["scale"]*parms["degree"]*x_aa[
   n]^(parms["degree"]-1)
k<-k+1
}
}

if(identical("tanhdot", kernel_function)){ #if hiperbolic
   tangent kernel
parms <- unlist(kparameters)
#Derivative
prodE <- apply(x,1,FUN=function(obs){x%*%obs}) #producto
   escalar
res <- (parms["scale"]*x[,i])/cosh(parms["scale"]*prodE+parms
   ["offset"])
#Metric Tensor
norma <- diag(prodE)
off_g <- parms["scale"]*norma+parms["offset"]
x_ab <- apply(x, 2, FUN = function(aa){aa*x})
g_ab <- -2*parms["scale"]*x_ab*tanh(off_g)/cosh(off_g)
for(n in 1:nrow(x)){
g_ab[k,i] <- g_ab[k,i] + 1/cosh(off_g[n])^2
k<-k+1
} #return the derivation for hiperbolic tangent
}
```

```
if(identical("wave", kernel_function)){ #if Wave kernel
parms <- unlist(kparameters)
#Derivative
Anorm <- as.matrix(dist(x, diag=TRUE, upper=TRUE))
Adif <- sapply(x[,vars[i]], FUN = function(aa){aa-x[,vars[i
   ]]})
s <- sin(Anorm/parms)
c<- cos(Anorm/parms)
der <- Adif*parms/Anorm^3 * s + Adif/Anorm^2 * c
#tensor
g_ab <- 1/(3*parms^2)
#Derivative*tensor
res <- der/g_ab
}#return the derivation for wave kernel

cl[[i]] <- res #vector with direction of maximum growth of
   variable 'i' for each sample data points
}

if(identical("polydot",kernel_function) || identical("tanhdot
   ", kernel_function))
cl <- derTensor(g_ab, cl, vars) #return the product
   derivative*metric tensor

return(cl)
}

##############################
#producte Derivative * tensor #
##############################

derTensor = function(tensor, cl, vars){
#Return the product form Derivative * tensor m tric
#g_ab: metric tensor
#cl: Derivative
#vars: variable
#Group by individual
g<-list()
der<-list()
g_ab <-tensor[,vars[1]]
for(i in 2:length(vars)) #delate the non-important variable
g_ab <- t(rbind(tensor[,vars[i]], g_ab[i]))

for(i in 1:nrow(cl[[1]])){ #add the first individuals
g[[i]]<- g_ab[i,]
der[[i]] <-cl[[1]][,i]
}
k<-1+nrow(cl[[1]])#build the correct matrix by individual x_0
```

```
for(m in 2:ncol(g_ab)){
for(i in 1:nrow(cl[[1]])){
g[[i]] <- rbind(g[[i]], g_ab[k,])
der[[i]] <- rbind(der[[i]], cl[[m]][,i])
k<-k+1
}
}
#Product
prod <- list()
for(i in 1:nrow(cl[[1]]))
prod[[i]] <- t(der[[i]])%*%t(solve(g[[i]]))

#ungroup by variable
res <- list()
for(i in 1:ncol(prod[[1]]))
res[[i]] <-prod[[1]][,i]
for(m in 2:nrow(prod[[1]])){
for(i in 1:ncol(prod[[1]]))
res[[i]] <- rbind(res[[i]], prod[[m]][,i])
}

return(res)
}

################################
######## Kernel Matrix #######
################################

#Hellinger Kernel

kernelMatrix_Hellinger=function(KM_x){
#Comput the Kernel Matrix in case it is the Hellinger Kernel
#Input:
# KM_x: data set indexed by row
res<- 4* apply(KM_x,1,FUN=function(obs){sqrt(KM_x)%*%sqrt(obs
   )})
return(res)
}

#Matriu Kernel Rational Quadratic

kernelMatrix_RatQuad=function(KM_x, k_parameters){
#comput the kernel Matrix for Rational Quadratic Kernel
#Input:
#   KM_x: data set indexed by row
#   k_parameters: list with kernel parameters
parms <- unlist(kparameters)
res <- 1-as.matrix((dist(KM_x, diag=TRUE, upper=TRUE)^2)/(
   dist(KM_x, diag=TRUE, upper=TRUE)^2+parms["sigma"]^2))
```

```
return(res)
}

#Matriu Multiquadratic Kernel

kernelMatrix_MultQuad = function(KM_x, kparameters){
#comput the kernel Matrix for Multiquadratic Kernel
#Input:
#   KM_x: data set indexed by row
#   k_parameters: list with kernel parameters
parms <- unlist(kparameters)
res <- sqrt(as.matrix(dist(KM_x, diag=TRUE, upper=TRUE)^2)+(
   parms["sigma"]^2))
return(res)
}

#Inverse Multiquadratic Kernel
kernelMatrix_InvMultQuad = function(KM_x, kparameters){
#comput the kernel Matrix for Rational Quadratic Kernel
#Input:
#   KM_x: data set indexed by row
#   k_parameters: list with kernel parameters
parms <- unlist(kparameters)
res <- 1/sqrt(as.matrix(dist(KM_x, diag=TRUE, upper=TRUE)^2)
   +(parms["sigma"]^2))
return(res)
}

#Chi-quadrat Kernel
kernelMatrix_ChiQuad = function(KM_x){
#comput the kernel Matrix for InverseMultiquadratic Kernel
#Input:
#   KM_x: data set indexed by row
s <- list() #sum for each 2 variables
p<-list() #product for each 2 variables
d <- list()
for(i in 1:ncol(x)){
s[[i]]<- sapply(KM_x[,i], FUN = function(aa){aa+KM_x[,i]})
p[[i]]<- sapply(KM_x[,i], FUN = function(aa){aa*KM_x[,i]})
d[[i]] <- p[[i]]/s[[i]]
}
for(i in 1:(ncol(x)-1)){
res <- d[[i]]+d[[i+1]]
}
return(2*res)
}

#Poisson kernel
kernelMatrix_poisson = function(KM_x){
```

```
#comput the kernel matrix for Poisson kernel
#Input:
#   KM_x: data set indexed by row
suma <- 0
Apos <- list()
for(i in 1:ncol(x)){
Apos[[i]] <- sapply(x[,vars[i]], FUN = function(aa){(sqrt(aa)
   -sqrt(x[,vars[i]]))^2})
}
for(i in 1:ncol(x)){
suma <- suma + Apos[[i]]
}
res <- 4*exp(-1/2 * suma)
}



gderivation=function(x, vars, kernel_function, kparameters,
   kernelM, coef){
#Comput the direction of maximum growth (derivation at s=0)
   of determined variables for each sample
#Input:
#       x: matrix indexed by rows or list of matrixs with row
    data
#       vars: variable names
#       kernel_function: kernel function
#       kparameters: list with kernel parameters
#       kernelM: kernel matrix
#       coef: linear combination coefficients
#Output: list with the direction of maximum growth (
   derivation at s=0) of determined variables for each sample
if(is.list(x)){ #when more than 1 dataset is given
cl <- vector()
for(j in 1:length(x)){ #for each dataset
lx <- x[[j]] #matrix of dataset 'j'
lvars <- vars[vars%in%colnames(lx)] #variables in lx
lkparameters <- kparameters[[j]] #kernel parameters for
   dataset 'j'
lkernelM <- kernelM[[j]] #kernel matrix for dataset 'j'
lkernel_function <- kernel_function[[j]] #kernel function for
    dataset 'j'
if(length(lvars)>0){ #if variable to represent is in dataset
   'j'
cl2 <- derivation(lx, lvars, lkernel_function, lkparameters,
   lkernelM) #Comput the direction of maximum growth (
   derivation at s=0) of determined variables for each sample
names(cl2) <- lvars #rename each element with the variable
   name
```

```
}
if(length(lvars)==0){ #no variable found in dataset 'j'
cl2 <- NULL #return NULL
}
cl <- c(cl,cl2) #list concatenation
}
}
if(is.matrix(x)){ #when 1 dataset is given
cl <- derivation(x, vars, kernel_function, kparameters,
   kernelM) #Comput the direction of maximum growth (
   derivation at s=0) of determined variables for each sample
names(cl) <- vars #rename each element with the variable name
}
return(cl) #return a list with the direction of maximum
   growth (derivation at s=0) of determined variables for
   each sample
}



kernelMatrix_computation=function(KM_kernel_function, KM_
   kparameters,KM_x){
#Comput the kernel matrix using the kernelMatrix function
   from krenlab package.
#Input:
#       KM_kernel_function: kernel function
#       KM_kparameters: list with kernel parameters
#       KM_x: data set indexed by row
#Output: Kernel matrix
if(identical("rbfdot",KM_kernel_function)){ #gaussian kernel
res <- kernelMatrix(rbfdot(unlist(KM_kparameters)),KM_x)
} #gaussian formula
if(identical("polydot",KM_kernel_function)){ #polinomial
   kernel
res <- kernelMatrix(polydot(unlist(KM_kparameters)),KM_x)
} #polynomial kernel

if(identical("anovadot",KM_kernel_function)){ #ANOVA kernel
res <- kernelMatrix(anovadot(unlist(KM_kparameters)),KM_x)
} #base radial ANOVA formula

if(identical("tanhdot", KM_kernel_function)){ #Hyperbolic
   tangent kernel
res <- kernelMatrix(tanhdot(unlist(KM_kparameters)), KM_x)
}

if(identical("vanilladot", KM_kernel_function)){ #linear
   kernel
res <- kernelMatrix(vanilladot(unlist(KM_kparameters)), KM_x)
```

```
}
if(identical("hellinger", KM_kernel_function)){
res <- as.kernelMatrix(kernelMatrix_Hellinger(KM_x))#
   hellinger kernel
}
if(identical("multquad", KM_kernel_function)){
res <- as.kernelMatrix(kernelMatrix_MultQuad(KM_x)) #
   multiquadratic kernel
}
if(identical("ratquad", KM_kernel_function)){
res <- as.kernelMatrix(kernelMatrix_RatQuad(KM_x)) #rational
   Quaratic Kernel
}
if(identical("invmultquad", KM_kernel_function)){
res <- as.kernelMatrix(kernelMatrix_InvMultQuad(KM_x)) #
   Inverse Multiquaratic Kernel
}
if(identical("chiquad", KM_kernel_function)){ #chi square
   kernel
res <- as.kernelMatrix(kernelMatrix_ChiQuad(KM_x))
}


return(res)
}



#------------------------------------------>>>> KPCAplus
   function

KPCAplus = function(x, kernel_function, kparameters, xpc=1,
   ypc=2, factors=NULL, variables=NULL, combination="+",  var
   _combination=NULL, coef_combination=NULL, scale=0.5, plot.
   kPCA=TRUE, plot.profile=FALSE, variables_discovery=NULL,
   discovery.col="orange", weights=NULL, text.sample=FALSE,
   show.legend=TRUE, ...){
#... -> refers to other kpca plot parameters


##############################################
# Parameters control                         #
##############################################

if(!is.matrix(x) & !is.list(x)) stop("'x' must be a matrix
   indexed by row\n")
if(!is.character(kernel_function) & !is.list(kernel_function)
   ) stop("'kernel_function' must be a character or a list of
    characters\n")
```

```
if(!is.list(kparameters)) stop("'kparameters' must be a list\
    n")
if(!is.numeric(xpc) | xpc>8 | xpc<=0) stop("'xpc' must be
    numeric, range 1-8.\n")
if(!is.numeric(ypc) | ypc>8 | xpc<=0) stop("'ypc' must be
    numeric, range 1-8.\n")
if(!is.null(factors) & !is.data.frame(factors)) stop("'
    factors' must be a dataframe\n")
if(!is.null(factors)) factors <- data.frame(apply(factors,2,
    FUN=function(ff){return(as.factor(ff))}))
if(!is.null(variables) & !is.character(variables)) stop("'
    variables' must be a character vector\n")
if(!is.character(combination)) stop("'combination' must be a
    character defining an operation: '+', '*'\n")
if(!is.null(var_combination) & !is.character(var_combination)
    ) stop("'var_combination' must be a character vector\n")
if(!is.null(coef_combination) & !is.numeric(coef_combination)
    ) stop("'coef_combination' must be a numeric vector\n")
if(!is.numeric(scale)) stop("'scale' must be numeric\n")
if(!is.logical(plot.kPCA)) stop("'plot.kPCA' must be logical\
    n")
if(!is.logical(plot.profile)) stop("'plot.profile' must be
    logical\n")
if(plot.profile & !is.null(variables) & is.null(factors))
    stop("'plot.profile' must be FALSE or 'factors' should be
    provided \n")
if(!is.null(variables_discovery) & !is.numeric(variables_
    discovery)) stop("'variables_discovery' must be a numeric
    vector\n")
if(!is.null(variables) | !is.null(var_combination)){
all_var <- c(variables, var_combination)
if(is.list(x)) data_var <- unlist(lapply(x,FUN=function(xx){
    return(colnames(xx))}))
if(is.matrix(x)) data_var <- colnames(x)
if(sum(all_var%in%data_var)!=length(all_var)) stop("'
    variables' or 'var_combination' sepcified are not in the
    data set provided\n")
if(!is.null(weights) & !is.numeric(weights)) stop("'weight'
    must be numeric\n")
if(is.numeric(weights) & !is.list(x)) stop("'weight' defined
    when only one data set used\n")
if(is.numeric(weights) & length(x)!=length(weights))stop("'
    weight' must have the same lenght as the number of
    datasets analyzed\n")
if(!is.character(combination)) stop("'combination' must be
    character. 'combination' defines the combination variables
     equation operation, being addition '+', multiplicative '*
    ', substraction '-' or division '\\'\n")
}
```

```
if(!is.logical(show.legend)) stop("'show.legend' must be
    logical\n")

kpca_plus_results <- list() #object with resulta to return

##############################################
# Compute KPCA                               #
##############################################

xpc <- as.integer(xpc) #integer
ypc <- as.integer(ypc) #integer

if(is.list(x)){ #when more than 1 dataset is given
nd <- length(x) #number of datasets
if(is.null(weights)) weights <- rep(1,length(x)) #weights of
    datasets

kernelM <- list() #list to save the kernel matrix for each
    dataset

for(i in 1:length(x)){  #compute kernel matrix for each
    dataset
kernelM[[i]] <- kernelMatrix_computation(KM_kernel_function=
    kernel_function[[i]], KM_kparameters=kparameters[[i]], KM_
    x=x[[i]])*weights[i]
kernel_matrix <- as.kernelMatrix(Reduce("+", kernelM)) #
    kernel matrix of combinated kernels (addition)
}
kpca_result <- kpca(as.kernelMatrix(Reduce("+", kernelM)),
    features=8) #kpca from combinated kernels (kernlab)
}

if(is.matrix(x)){ #when 1 dataset is given
nd <- 1 #number of datasets
kernelM <- kernelMatrix_computation(KM_kernel_function=kernel
    _function, KM_kparameters=kparameters, KM_x=x) #compute
    kernel matrix for dataset
kernel_matrix <- kernelM #kernel matrix
if(identical(kernel_function, "vanilladot") || identical(
    kernel_function, "polydot") || identical(kernel_function,
    "rbfdot") || identical(kernel_function, "tanhdot") ){
kpca_result <- kpca(x, kernel=kernel_function, kpar=
    kparameters, features=8, ...) #kpca from given data (
    kernlab)
} else{
kpca_result <- kpca(kernel_matrix, features=8, ...) #kpca
    from given data (kernlab)
}
}
```

```
kpca_plus_results$datasets <- list(number_data_sets=nd, data=
   x) #list with numer of datasets and raw datasets
kpca_plus_results$KPCA <- list(weights=weights, kernel=kernel
   _function, kernel_parameters=kparameters, kernel_matrix=
   kernel_matrix, kpca=kpca_result) #list with weights,
   kernel function, kernel parameters, kernel matrix and
   output of kpca.

if(plot.kPCA==TRUE){ #visualizing the KPCA in a biplot

par()
if(show.legend==TRUE) par(mar=c(5.1, 4.1, 4.1, 8.1), xpd=TRUE
   )

limi <- max(abs(min(rotated(kpca_result)[,xpc],rotated(kpca_
   result)[,ypc])), max(rotated(kpca_result)[,xpc],rotated(
   kpca_result)[,ypc]))
inferior_lim <- (-1*limi) - 1
superior_lim <- limi + 1

if(!is.null(factors)){ #factors given
col_kpca <- as.numeric(factors[,1]) #assign colors to first
   factor
pch_kpca <- 19
if(ncol(factors)>1){ #more than 1 factor given
pch_kpca <- as.numeric(factors[,2])-1 #assign points symbols
   to second factor
}

plot(rotated(kpca_result)[,c(xpc,ypc)], col=col_kpca, pch=pch
   _kpca, xlab=paste("Principal Component", xpc, sep=" "),
   ylab=paste("Principal Component", ypc, sep=" "), las=1,
   xlim=c(inferior_lim, superior_lim), ylim=c(inferior_lim,
   superior_lim), ...) #biplot KPCA

if(show.legend==TRUE){
legend("topright", inset=c(-0.2, 0), legend=levels(factors
   [,1]), pch=15, col=1:length(levels(factors[,1])), title=
   names(factors)[1]) #legend first factor
if(ncol(factors)>1){ #more than 1 factor given
legend("right", inset=c(-0.2,0), legend=levels(factors[,2]),
   pch=0:(length(levels(factors[,2]))-1), title=names(factors
   )[2]) #legend second factor
}
}
}
```

```
if(is.null(factors)){ #no factors given
col_kpca <- "black" #color points
pch_kpca <- 19 #symbol points
plot(rotated(kpca_result)[,c(xpc,ypc)], col=col_kpca, pch=pch
   _kpca, xlab=paste("Principal Component", xpc, sep=" "),
   ylab=paste("Principal Component", ypc, sep=" "), las=1,
   xlim=c(inferior_lim, superior_lim), ylim=c(inferior_lim,
   superior_lim), ...) #biplot KPCA
}

if(text.sample==TRUE){ #if TRUE print sample names
if(is.list(x)) snames <- rownames(x[[1]]) #sample names when
   more than 1 dataset is given
if(is.matrix(x)) snames <- rownames(x) #sample names when 1
   dataset is given
text(rotated(kpca_result)[,c(xpc,ypc)], labels=snames, cex
   =0.6, adj=1) #print sample names to KPCA biplot
}

abline(v=0, lty=3, xpd=FALSE) #add a vertical dotted line at
   0
abline(h=0, lty=3, xpd=FALSE) #add an horitzontal dotted line
    at 0

}

##############################################
# Represent original variables               #
##############################################

#origin point for each sample in the kpca representation
x0 <- rotated(kpca_result)[,xpc] #projection points of PC x-
   axis
y0 <- rotated(kpca_result)[,ypc] #projection points of PC y-
   axis

Vcuc <- pcv(kpca_result)[,c(xpc,ypc)] #matrix with principal
   components
MM <- (diag(nrow=length(x0))) - (matrix(1/length(x0), ncol=
   length(x0), nrow=length(x0))) #part of the KPCA images
   projection formula



#-----> Linear combination of original variables
   representation

if(!is.null(var_combination)){ # variables for combination
   defined
```

```
if(length(var_combination)==1) stop("'var_combination' must
   be a character vector of length >1\n") #we need more than
   one variable defined in order to do de variable
   combination
if(!is.null(coef_combination) & length(coef_combination)!=
   length(var_combination)) stop("'var_combination' and 'coef
   _combination' must have the same length\n") #verify that
   the number of coeficients, if defined, are the same as
   number of variables
if(is.null(coef_combination)) coef_combination <- rep(1,
   length(var_combination))
cl <- gderivation(x, var_combination, kernel_function,
   kparameters, kernelM) #kernel derivation as part of the
   KPCA images projection formula
for(vc in 1:length(cl))
cl[[vc]] <- cl[[vc]]*coef_combination
res <- Reduce(combination, cl) #linear combination
x1 <- res %*% MM %*% Vcuc #images projection in KPCA
norm_x1 <- apply(x1,1,FUN=function(g){sqrt(sum(g^2))}) #
   variable standarization
x0n <- (x1/norm_x1)*scale #reduce arrow length
if(plot.kPCA==TRUE) arrows(x0, y0, x1=(x0+x0n[,1]), y1=(y0+
   x0n[,2]), col="black", length=0.10, lty=1, lwd=1) #plot
   the arrow pointing the direction of maximum growth
comvar <- var_combination[1] #variables for legend
for(i in 2:length(var_combination)){ #if more than one
   variable, define legend
comvar <- paste(comvar, var_combination[i], sep=combination)
   #variable legend
}
if(is.null(variables) & plot.kPCA==TRUE & show.legend==TRUE)
   legend("bottomright", inset=c(-0.2,0), legend=comvar, pch=
   c("-"),col="black",title="Variables", bg="white") #print
   variable combination legend
}



#-----> Original variables representation

if(!is.null(variables)){ # variables defined
cl <- gderivation(x, variables, kernel_function, kparameters,
    kernelM) #kernel derivation as part of the KPCA images
   projection formula
for(i in 1:length(variables)){ #for each variable
res <- cl[[i]] #get the maximum growth for variable 'i' for
   each sample
x1 <- res %*% MM %*% Vcuc #images projection in KPCA
norm_x1 <- apply(x1,1,FUN=function(g){sqrt(sum(g^2))}) #
```

```
    variable standarization
x0n <- (x1/norm_x1)*scale #reduce arrow length
if(plot.kPCA==TRUE) arrows(x0, y0, x1=(x0+x0n[,1]), y1=(y0+
    x0n[,2]), col=i+1, length=0.10, lty=1, lwd=1) #plot the
    arrow pointing the direction of maximum growth
}
if(!is.null(var_combination) & plot.kPCA==TRUE & show.legend
    ==TRUE) legend("bottomright", inset=c(-0.2,0), legend=c(
    comvar,names(cl)), pch=c("-"),col=c(1:(length(variables)
    +1)),title="Variables", bg="white") #plot legend with
    variable combination and individual variables
if(is.null(var_combination) & plot.kPCA==TRUE & show.legend==
    TRUE) legend("bottomright", inset=c(-0.2,0), legend=names(
    cl), pch=c("-"),col=c(2:(length(variables)+1)),title="
    Variables", bg="white") #plot legend with individual
    variables


#plot variable profile
if(plot.profile){
par(mar=c(6.1, 4.1, 4.1, 8.1), xpd=TRUE)
if(is.list(x)){
for(k in 1:length(x)){
xx <- x[[k]]
vars <- variables[variables%in%colnames(xx)]
if(length(vars)>0){
for(i in 1:length(vars)){
if(ncol(factors)==1){
stripchart(xx[,vars[i]] ~ factors[,1], method = "jitter",
    jitter=0.05,  main=paste(vars[i],"profile", sep=" "), col=
    "black", vertical= TRUE, pch=19, ylab=paste("Variable",
    variables[i],sep=" "), xlab=names(factors)[1], las=1)
points(1:length((levels(factors[,1]))), tapply(xx[,vars[i]],
    factors[,1],median), col="red", type="b", pch=15)
}

if(ncol(factors)>1){
stripchart(xx[,vars[i]] ~ factors[,1], method = "jitter",
    jitter=0.05,  main=paste(vars[i],"profile", sep=" "), col=
    as.numeric(factors[,2]), vertical= TRUE, pch=NA, ylab=
    paste("Variable",vars[i],sep=" "), xlab=names(factors)[1],
     las=1)
points(jitter(as.numeric(factors[,1]),0.2), xx[,variables[i
    ]], col=as.numeric(factors[,2]), pch=19)
for(j in 1:length(levels(factors[,2]))){
sel <- which(factors[,2]==levels(factors[,2])[j])
points(1:length((levels(factors[,1]))), tapply(xx[sel,vars[i
    ]],factors[sel,1],median), col=j, type="b", pch=15)
}
```

```
legend("topright", inset=c(-0.2,0), legend=levels(factors
   [,2]), pch=19, col=1:length(levels(factors[,2])), title=
   names(factors)[2])
}
}
}
}
}

if(is.matrix(x)){
for(i in 1:length(variables)){
if(ncol(factors)==1){
stripchart(x[,variables[i]] ~ factors[,1], method = "jitter",
    jitter=0.05, main=paste(variables[i],"profile", sep=" "),
    col="black", vertical= TRUE, pch=19, ylab=paste("Variable
   ",variables[i],sep=" "), xlab=names(factors)[1], las=1)
points(1:length((levels(factors[,1]))), tapply(x[,variables[i
   ]],factors[,1],median), col="red", type="b", pch=15)
}

if(ncol(factors)>1){
stripchart(x[,variables[i]] ~ factors[,1], method = "jitter",
    jitter=0.05, main=paste(variables[i],"profile", sep=" "),
    col=as.numeric(factors[,2]), vertical= TRUE, pch=NA, ylab
   =paste("Variable",variables[i],sep=" "), xlab=names(
   factors)[1], las=1)
points(jitter(as.numeric(factors[,1]),0.2), x[,variables[i]],
    col=as.numeric(factors[,2]), pch=19)
for(j in 1:length(levels(factors[,2]))){
sel <- which(factors[,2]==levels(factors[,2])[j])
points(1:length((levels(factors[,1]))), tapply(x[sel,
   variables[i]],factors[sel,1],median), col=j, type="b", pch
   =15)
}
legend("topright", inset=c(-0.2,0), legend=levels(factors
   [,2]), pch=19, col=1:length(levels(factors[,2])), title=
   names(factors)[2])
}
}
}
}
}



##############################################
# Variables discovery                        #
##############################################
```

```
if(!is.null(variables_discovery)){ #done variable discovery
if(plot.kPCA==TRUE){
arrows(x0=variables_discovery[1], y0=variables_discovery[2],
   x1=variables_discovery[3], y1=variables_discovery[4], col=
   discovery.col, length=0.10, lty=1, lwd=3)} #print the
   arrow showing the direction of interest in the KPCA
if(is.matrix(x)){ #when one dataset transform all the input
   objects as list.
x <- list(x)
kernel_function <- list(kernel_function)
kparameters <- list(kparameters)
kernelM <- list(kernelM)
}

cl2 <- vector()
vars <- vector()
dataset <- vector()
for(k in 1:length(x)){ #for each dataset
variables <- colnames(x[[k]]) #variables of dataset 'k'
cl <- gderivation(x[[k]], variables, kernel_function[[k]],
   kparameters[[k]], kernelM[[k]]) #kernel derivation as part
    of the KPCA images projection formula. Give the direction
    of maximum growth for each variable in each sample
scl2 <- list()
for(i in 1:length(variables)){ #for each variable
res <- cl[[i]] #derivation variable 'i'
x1 <- res %*% MM %*% Vcuc #images projection in KPCA for
   varaible 'i'
scl2[[i]] <- x1
}
cl2 <- c(cl2,scl2) #points
vars <- c(vars,variables) #variables
dataset <- c(dataset,rep(k,length(variables))) #dataset
}


norm_discovery <- variables_discovery[3:4]/sqrt(sum(variables
   _discovery[3:4]^2))
alignement <-lapply(cl2,FUN=function(dd){
if(is.matrix(dd)){ rdd <- dd%*%norm_discovery / apply(dd,1,
   FUN=function(g){sqrt(sum(g^2))})}
else { rdd <- dd*norm_discovery / sqrt(sum(dd^2))}
return(rdd)})

sd.l    <- unlist(lapply(alignement,sd))
mean.l <-unlist(lapply(alignement,mean))
alignement_result <- data.frame(names = vars, dataset =
   dataset, alin.mean = mean.l, alin.sd = sd.l)
```

```
kpca_plus_results$variable_discovery <- alignement_result
}

return(kpca_plus_results)

} #end function
```