Manual d'usuari CBR, PCA i Dades

Llorenç Burgas Nadal

Índex

Índex de figures

Part I

Introducció

1 Motivació

Aquest document és un manual d'usuari per a qui hagi d'utilitzar aquest codi posteriorment. S'intentarà descriure les funcions i objectes que aporta el codi en qüestió de la forma més clara i entenedora possible. Aquí no s'explicaran els fonaments bàsics de PCA (*principal component analysis*) ni de CBR (*Case-based reasoning*). Les explicacions es limitaran a aportar tots els coneixements que necessitarà un usuari per crear una aplicació utilitzant aquestes "Llibreries".

2 Requisits

Les classes que es proporcionen (Dades, PCA i CBR) estan creades amb Matlab®2010a. S'ha aprofitat que en les darreres versions del producte s'-ha incorporat la possibilitat de programar amb Orientació a Objectes (OO). Programar amb OO en Matlab®ens aporta avantatges importants a l'hora d'encapsular el codi i determinar els permisos d'accés als atributs dels nostres objectes. Com a inconvenients trobem que necessitarem una versió del Matlab®2008a o posterior.

Respecte als requisits mínims del sistema no necessitem res més que un equip capaç d'executar el Matlab®. tot i això, és molt recomanable disposar d'un

equip superior als mínims establerts per tal d'assegurar una bona resposta. Els requisits mínims per poder executar el Matlab®2008a són:

CPU: Pentium®4 o AMD Athlon 64

Espai al disc dur: 510 MB lliures per instal·lar el Matlab®

RAM: 512MB

Sistema operatiu: Windows®XP , Devian 4.0, Fedora Core 4, Red hat Enterprise Linux v.4 , Mac®OS X 10.4

Tarja gràfica: 16 bits compatible amb OpenGL

Per més informació sobre els requisits mínims del Matlab®i les diferents versions disponibles es pot visitar la seva pagina web:
http://www.mathworks.com/products/matlab/requirements.html

3 Descripció General

S'han definit 3 objectes:

- **Dades** - Serà la nostra estructura bàsica per guardar tota la informació necessària en memòria.
- **PCA** - Ens aportarà un seguit de funcions que ens permetran realitzar un anàlisi de components principals (*principal component analysis*) sobre un objecte de tipus Dades.

- **CBR** - Permetrà aplicar les funcions bàsiques de raonament basat en casos (*case-based reasoning*) treballant sobre objectes de tipus Dades i PCA.

Tant PCA com CBR actuen sobre els objectes de tipus Dades, obtenint les dependències entre les classes que es mostren a la Figura1. Tanmateix aquestes classes s'han dissenyat com a llibreries, per això no disposen d'una interface d'usuari i moltes de les funcions poden resultar pesades d'utilitzar manualment.

Important:
En Matlab®quan es treballa amb objectes es passa sempre per paràmetre l'objecte al què ens estem referint. Així doncs, la crida x.get() passarà a invocar la funció get(x) de manera automàtica. En les descripcions de totes les funcions dels objectes ometrem aquest paràmetre ja que és simple notació del llenguatge Matlab®.

4 UML

A nivell global podem observar que les 3 classes que expliquem a continuació tenen relacions entre elles. En la Figura1 podem observar les dependències existents entre les diferents classes implementades. De la Figura1 s'extreu, per una banda, que un objecte PCA no pot existir sense un objecte Dades. Al mateix temps, un objecte CBR necessita de l'existència alhora d'un objecte Dades i un altre de PCA.

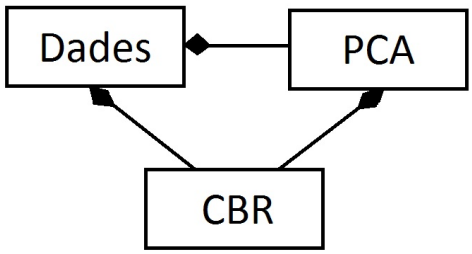


Figura 1: Diagrama de Classes

5 Notació que utilitzarem

En aquest apartat explicarem les diferents abreviatures que hem utilitzat en la redacció d'aquesta documentació. A més a més, també s'hi expliquen les característiques del Matlab®que hem utilitzat durant la implementació.

Abreviatures:

PCA: Anàlisi de components principals (*principal component analysis*).

CBR: Raonament basat en casos (*case-based reasoning*)

Lot o Batch: Conjunt d'observacions preses al llarg del temps d'un mateix procés, i que es repeteix sobre altres individus de la mateixa naturalesa.

Null: Quan parlem de null o valor null en Marlab®ens referím a conjunt buit, que en Matlab®es representa amb [].

dim: És l'abreviació de dimensió i normalment s'aplica sobre vectors i matrius.

Handle: Qualsevol tipus de dades vàlid pel Marlab®.

Index: Qualsevol tipus de indexació vàlida pel Marlab®.

BW: Desdoblament en batch-wise d'una matriu tridimensional com la representada en la Figura4. Consisteix en mantenir com a files els lots i desdoblar com a columnes les variables al llarg del temps tal com es mostra a la Figura2. El format que utilitzarem serà:

dim 1 = Batch

dim 2 = Variables x Temps

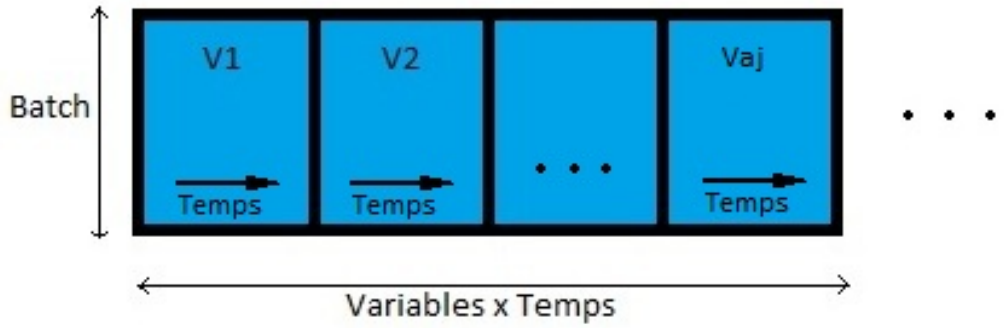


Figura 2: DadesBW

VW: Desdoblament en variable-wise de la matriu tridimensional representada en la Figura4. En aquest cas, a les columnes es mantenen les variables mesurades pels diferents lots, mentre que les files de la matriu bidimensional es desdoblen els diversos lots al llarg del temps, tal com es mostra a la Figura3. El format que utilitzarem serà:

dim 1 = Batch x Temps

dim 2 = Variables

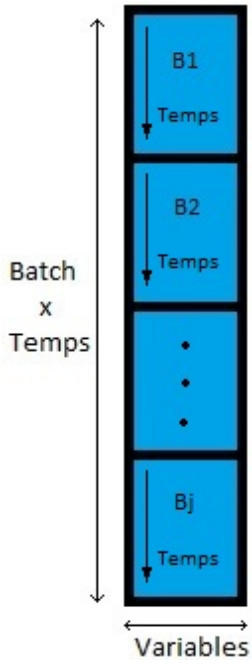


Figura 3: DadesVW

Peculiaritats del Matlab®:

Les dimensions de les dades en Matlab®: Matlab®utilitza la seva pròpia notació per referir-se a les diferents dimensions dels vectors i matrius (dimensió 1 = Abscisses, dimensió 2 = Ordenades, dimensió 3 = Profunditat ...). La Figura4 presenta un exemple d'aquesta notació per a una matriu tridimensional.

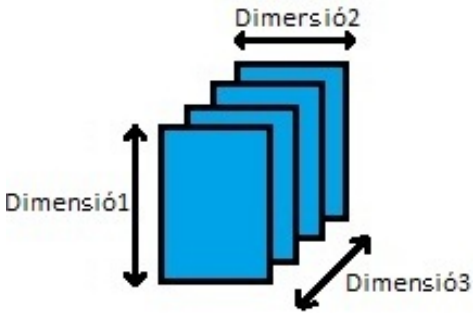


Figura 4: Dimensions en Matlab®

Retorn múltiple: El Matlab®permet fer retorn múltiple. En aquelles funcions que l'utilitzen, amb l'objectiu de facilitar-ne la comprensió s'ha utilitzat la següent notació Struct$[Tipus_1, Tipus_2, ...Tipus_N]$.

Part II

Dades

6 Descripció

La classe Dades és l'encarregada de muntar tota l'estructura de dades necessària per aplicar PCA i posteriorment, si es requereix, CBR. Dins d'aquest objecte, podem trobar, des de les dades originals guardades en una matriu 3D fins als mètodes necessaris per passar-les a 2D utilitzant els següents mètodes de desdoblament: bacth-wise i variable-wise.

Les dades més importants emmagatzemades en aquest objecte són:

- El nom del procés emmagatzemat a l'objecte actual.
- Informació general sobre les dades.
- Informació sobre les diferents fases.
- Informació sobre els lots de dades que tenim.
- Les dades del procés.
- La classificació dels diferents lots.

7 UML

La Figura5 mostra la representació utilitzant la notació UML de la classe Dades. Les files en vermell indiquen les funcions i atributs privats. Les funcions marcades amb blau són els constructors de a classe. Finalment les funcions marcades amb color verd són els mètodes amb accés públic visible.

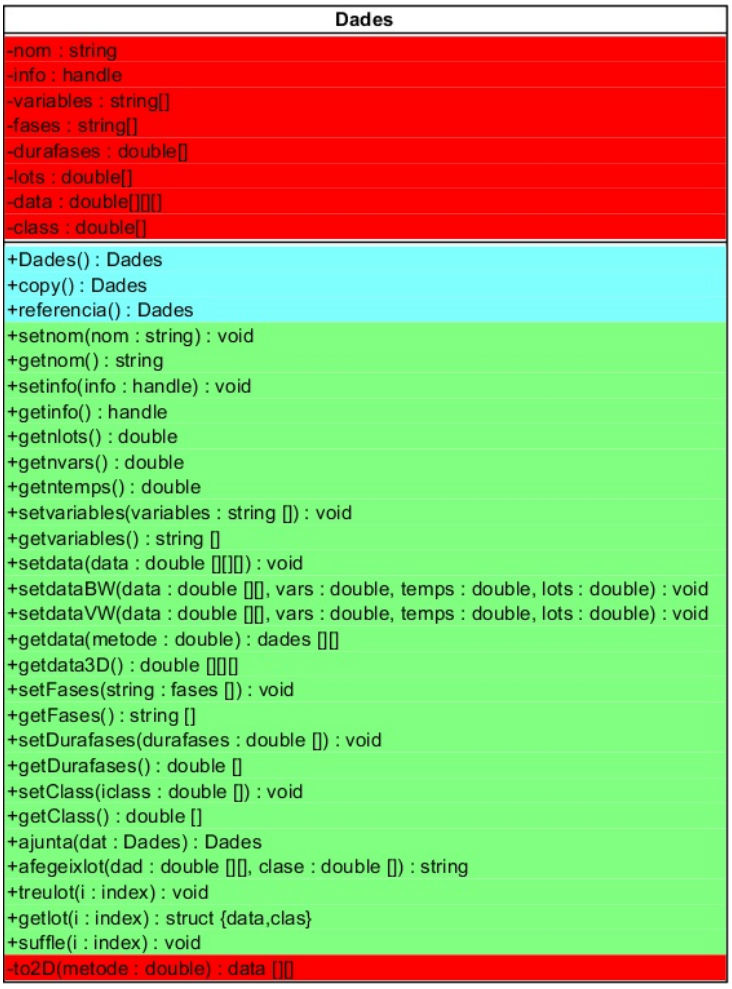


Figura 5: Classe Dades

La funcionalitat bàsica de la classe es emmagatzemar les dades que l'usuari ens entri. Els objectes d'aquesta classe seran els encarregats, doncs, de mantenir la informació en memòria i, a més a més, ens oferiran funcions per recuperar-les en diversos formats.

8 Utilització

En aquest apartat representarem un seguit d'exemples pràctics per mostrar el funcionament bàsic de la classe. Addicionalment, per tots els exemples presentats al llarg d'aquesta secció es treballarà amb conjunts de dades prèviament introduïts al Matlab®i emmagatzemats en fitxers ".mat".

Exemple 1

En aquest exemple utilitzarem les funcions que ens permeten modificar el contingut de l'objecte dades. Abans de tot, netejarem l'espai de treball (Workspace) de Matlab®per garantir que estem treballant amb les dades pertinents. A continuació, carreguem les dues matrius que contenen la informació amb que treballarem:

- Matriu_Good_Cross_Learning_1_Sort.mat : Aquest fitxer conté les dades de 59 lots d'un procés de depuració d'aigua sense cap mena de problema emmagatzemades en BW. Per cada lot s'han mesurat 4 variables durant 461 instants de temps.

- DatosProceso.mat : aquest fitxer conté informació diversa de cada lot (els noms de les variables mesurades, l'identificador de cada lot, el nom de les diverses fases, etc.).

El codi Matlab®que realitza aquesta part del codi es mostra en la Figura6.

```
%netejem el workspace
clear all;
clc;

%entrem les dades
load Matriu_Good_Cross_Learning_1_Sort.mat;
load 'DatosProceso.mat';
```

Figura 6: Codi Matlab®on entrem les dades a utilitzar al workspace

Seguidament creem un objecte de tipus Dades, utilitzant el constructor de la classe, per després introduir les Dades de les matrius carregades. La Figura7 representa les instruccions necessàries per crear l'objecte dades "dat" i la seva posterior utilització. S'ha utilitzat la funció "setdataBW" perquè les dades del procés estan guardades al .mat en aquest format i se li han passat per paràmetre les característiques de les dades com necessita Si les dades haguessin estat emmagatzemades en 3D, llavors s'hagués usat la funció "setdata", mentre que per dades en format VW s'empra la funció "setdataVW". Finalment, la classe també permet copiar un objecte dades a un altre de prèviament creat.

Un cop entrades totes les dades a la nostra instància de l'objecte Dades "dat" procedirem a explorar les diverses formes de recuperar les dades de la

```
%creem un objecte Dades buit
dat=Dades();
%li assignem dades
dat.setdataBW(MatriuGCL1sort,4,461,59);
%al .mat estan en batch-wise
%posem els noms de les variables del proces
dat.setvariables(model_process_data.var_name);
%fixem la duracio de les fases
dat.setdurafases(str2double(model_process_data.phase_time));
%donem nom a les fases
dat.setfases(model_process_data.phase_name);
%assignem les classes son totes de la mateixa clase
dat.setclass(ones(1,59));
```

Figura 7: Codi Matlab®per crear i inicialitzar un objecte Dades

instància "dat":

- Matriu tridimensional (getdata3D), que en aquest cas serà una matriu de 461 instants de temps×4 variables ×59 lots.
- Matriu bidimensional desdoblada en VW (getdata(0)) que en quest cas retornarà una matriu de 59 files(lots) i 461×4 columnes (Variables per temps)
- Matriu bidimensional desdoblada en VW (getdata(1)) que retornarà

una matriu bidimensional de 461×59 files (lots per instants de temps) i 4 columnes (variables).

```
%recuperem les dades en 3D
dadesen3D=dat.getdata3D();

%recuperem les dades en BW
dadesBW=dat.getdata(0);

%recuperem les dades en VW
dadesVW=dat.getdata(1);
```

Figura 8: Codi Matlab®diferents maneres de recuperar les dades

El codi Matlab®per obtenir cadascuna d'aquestes matrius es representa en la Figura8. Finalment el codi per accedir a un determinat lot (getlot()), eliminar-lo(treulot()) i afegir-lo (afegeixlot()) es mostra en la Figura9.

```
%Recuperem el lot 5 de les dades i la seva classe
[lot classe]=dat.getlot(5);

%eliminem el lot 5 de les dades
dat.treulot(5);

%ara el tornem a afegir(sempre s'afegeix al final)
dat.afegeixlot(lot, classe);
```

Figura 9: Codi Matlab®diferents maneres treballar amb dades

Exemple 2

En l'Exemple 1 hem mostrat les opcions per introduir i accedir a les diferents representacions de les dades. En aquest segon exemple farem un exemple més pràctic de la manipulació de l'estructura de dades. Per tal d'il·lustrar aquesta situació plantejarem la següent situació:

Suposem que ens arriben unes dades en format 3D. Ens diuen que són totes dades de processos bons però que tenen la sospita que se n'ha colat algun de dolent. Ens demanen que la localitzem i l'eliminem. Per fer-ho haurem de mostrar les dades del primer lot (que sabem segur que es bo) i comparar-les amb les de tots els altres i que un operari ens confirmi si són prou iguals o no.

Primer de tot haurem de saber com estan posades les dades que rebem. En el nostre cas tenim les dades en un .mat i a les dimensions hi tenim dim 1 = Temps, dim 2 = Variables, dim 3 = lots.Si mirem el help de la funció "setdata" en el Matlab®o bé la descripció d'aquesta a l'Apèndix d'aquest manual veurem que aquesta funció ens servirà per entrar les dades a l'objecte Dades sense haver de modificar-les. Vist això, començarem creant un objecte Dades. Per fer-ho utilitzarem el constructor. Tot seguit entrarem les dades en format 3D a l'objecte. A la Figura10 hi tenim el codi corresponent.

```
%carreguem les dades al workspace
load('dades3D.mat');
%creem un nou objecte dades
meu=Dades();
%li inserim la matriu 3D que ens donen
meu.setdata(dadesen3D);
%li insertem els noms de les variables
meu.setvariables('pH' 'O2' 'RedOx' 'Temp')
%li afegim la duració de les fases
meu.setdurafases([15 138 187 6 75 40])
%li afegim els noms de les fases
meu.setfases('F1' 'ANA' 'AE1' 'F2' 'ANO' 'AE2')
%afegim la classe dels lots
meu.setclass(ones(1,59));
eliminar=logical(meu.getnlots());%dimencionem el vector eliminar
eliminar(1)=0;%el primer no l'eliminarem
for i=2: meu.getnlots()
    figure(1);plot(meu.getlot(1));%mostrem els primer lots
    figure(2);plot(meu.getlot(i));%mostrem el lots a comparar
    resposta=input('S'ha d'eliminar?( 1=si 0=no)');
    eliminar(i)=(resposta==1);%afegim la resposta a on toca
end
meu.treulot(eliminar);%eliminem els que ens ha dit l'usuari
```

Figura 10: Codi de Matlab®amb un exemple d'utilització d la classe Dades

A la Figura10 podem observar que per recuperar lots hem utilitzat la funció " getlot()" aquesta funció ens retorna el lot o lots que li demanem per parametre. per eliminar els lots seleccionats hem utilitzat "treulot()" aquesta funció treurà els lots seleccionats, la seva classificació i actualitzara el numero de lots.

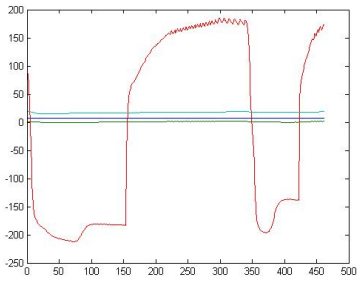


Figura 11: Dades Exemple 2 plot 1

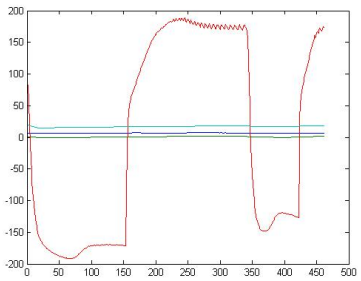


Figura 12: Dades Exemple 2 plot 2

A l'usuari li mostrarem un plot fixe com el de la Figura11 i un que canviara a cada iteració com el de la Figura12

Exemple 3:

En aquest darrer exemple d'utilització de la classe Dades el que farem serà il·lustrar el funcionament de les funcions de l'objecte dedicades a treballar amb fitxers.

Per guardar i carregar objectes Dades des de fitxer disposem dels mètodes "load" i "store". Aquests mètodes són simples d'utilitzar, com s'explica a l'annex disposen de 2 paràmetres un d'ells ens indica el nom del fitxer amb el qual estem treballant i l'altre el format de les Dades. Actualment es permeten 2 formats per les dades "mat" i "csv" el format mat és l'estàndard de Matlab®. En canvi el format "csv" és un format propi per la classe Dades.

Estructura del format CSV:

Nom, 'El nom del proces'
Variables, 'Numero de variables'
'El nom de les variables separades per comes'
Fases, 'numero de fases'
'el nom de les fases separades per comes'
Duracio de Fases, 'numero de fases'
'la duracio de les diferents fases separades pe comes'
ref. Lots, 'numero de lots'
'el nom de cada un dels lots'
class, 'numero de lots'

'la classe de cada un dels lots'

data, 'instants de temps'**,** 'numero de variables'**,** 'numero de lots'

'les dades en format Variable - Wise'

Primer de tot cal comentar que utilitzarem l'objecte Dades anomenat "meu" que hem creat en l'exemple anterior. El que farem serà guardar l'objecte en un fitxer que anomenarem "meu.txt" en format "csv" i posteriorment crearem un objecte i l'inicialitzarem llegint el fitxer que acabem de crear. A la Figura13 trobarem el codi comentat.

```
%guardem l'objecte 'meu' al fitxer 'meu.txt'
meu.save('meu.txt','csv');
%creem un object Dades nou anomenat 'd'
d=Dades();
%inicialitzem els camps de 'd' amb l'informació del fitxer 'meu.txt'
d.load('meu.txt','csv');
%guardem l'informació de 'd' a un nou fitxer per comprovar que
%s'ha inicialitzat correctament
d.save('meu2.txt','csv');
```

Figura 13: Codi de Matlab®amb un exemple d'utilització d la classe Dades

Part III

PCA

9 Descripció

La classe PCA serà l'encarregada de mantenir en memòria les dades necessàries per fer l'anàlisis de components principals. Aquesta classe també permet la projecció de dades sobre el model PCA emmagatzemat a més d'altres funcions útils. A més a més, i tal com es comentava a la secció 1, aquesta classe requereix treballar amb objectes de tipus Dades.

10 UML

El diagrama de classes de PCA es mostra en la Figura14, on s'han representat les funcions i atributs privats de color vermell, els constructors de color blau, i els mètodes públics de color verd.
Si ens fixem en el diagrama de classes de la Figura14 veiem que aquesta classe és bastant més extensa que la classe Dades. Aquesta classe també podem observar que hi ha una part dels mètodes accessibles a l'usuari (en verd a la Figura14) i una altre part no gens menyspreable de mètodes privats (en vermell a la Figura14).

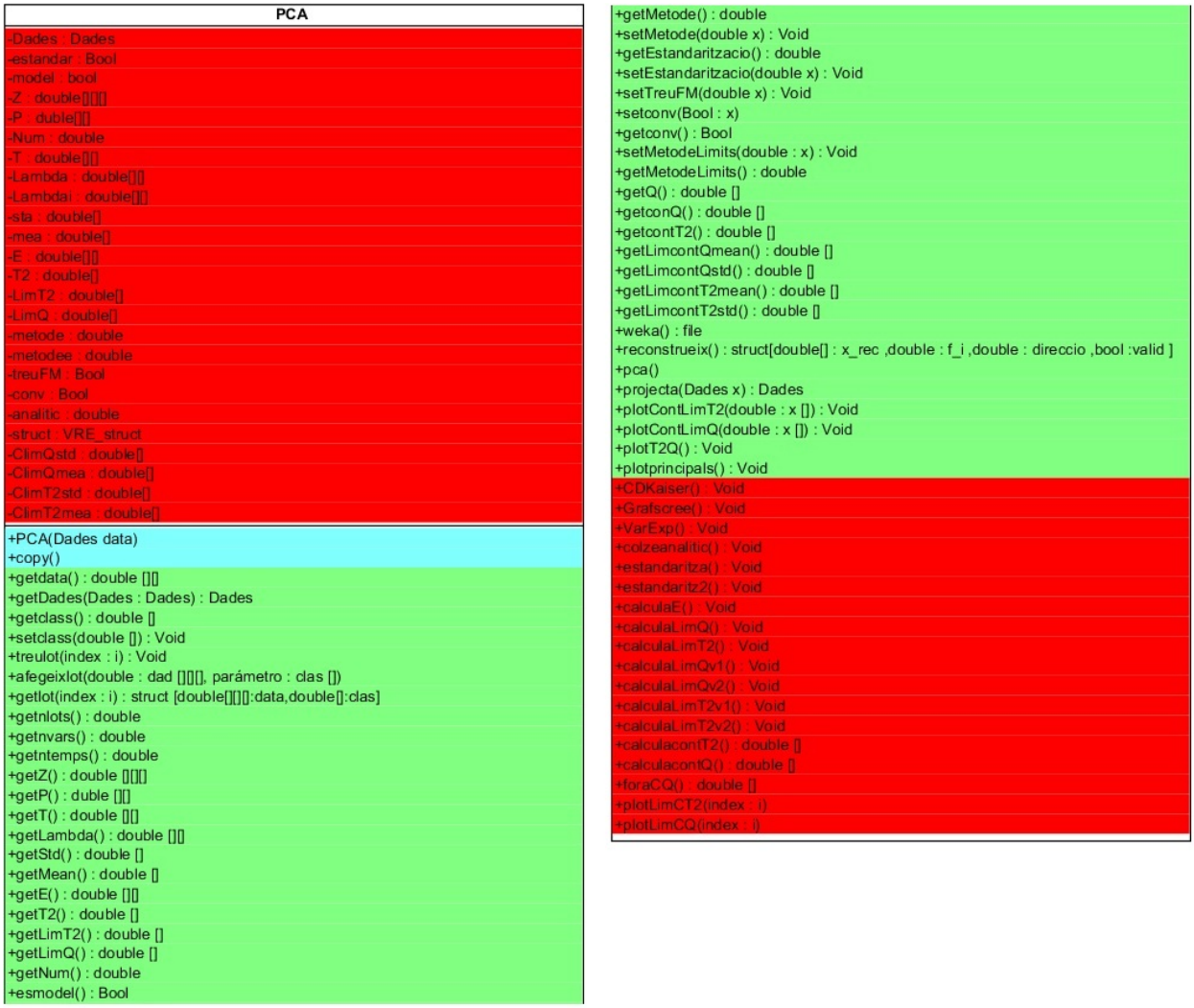


Figura 14: Classe PCA

11 Utilització

Com que el constructor de la classe PCA necessita que li passem un objecte de tipus Dades, el primer pas serà crear-ne un(en aquest cas agafem el mateix conjunt de dades de l'Exemple 1 concretament la Figura7). Un cop creat l'objecte Dades ja podem utilitzar el constructor de la classe PCA. En el nostre cas, crearem un model PCA VW on s'autoescalaran les dades per crear el model(removent la forma mitja del procés). Finalment el mètode de selecció de components principals serà VRE i els límits dels índexs de control els calcularem utilitzant les seves funcions de distribució. El codi per indicar totes aquestes opcions es mostra a la Figura15.

```
%creem un nou objecte PCA
MY_PCA=PCA(meu);
%1=VW,0=BW
MY_PCA.setconv(1);
%1=AS,2=GS,3=CS,4=BS
MY_PCA.setEstandaritzacio(1);
%0=Manual,1=Kaiser,2=colze,3=colze+punt,4=%lambda,5=VRA
MY_PCA.setMetode(5);
%1=analític,0=3sigma
MY_PCA.setMetodeLimits(1);
%1=treure,0=no treure
MY_PCA.setTreuFM(1);
```

Figura 15: Codi Matlab®que crea un nou objecte PCA

Les funcions que tenim a la nostra disposició les trobarem comentades a l'apèndix d'aquest document(Apèndix II-PCA). Però a continuació es mostrara la crida a les principals funcions assocciades a la manipulació d'un objecte (model) pca.a la Figura16 mostrarem com s'utilitzen algunes d'elles.

```
%agafem les dades que necessitem de l'objecte
p=MY_PCA.getP();
z=MY_PCA.getZ();
std=MY_PCA.getStd();
mea=MY_PCA.getmean();
t=MY_PCA.getT();
t2=MY_PCA.getLimT2();
lambda=MY_PCA.getLambda();
e=MY_PCA.getE();
```

Figura 16: Codi Matlab®exemple d'alguns getters de PCA

Un cop introduït el funcionament bàsic de la classe, ara passarem a realitzar un exemple pràctic de la classe.

Exemple Practic:

Tenim 2 grups de dades. Un és un grup de dades que sabem que són bones, i un altre grup que necessita ser comprovat. el que haurem de fer es crear un objecte PCA "model" per les que sabem que són bones i projectar-hi la resta. Un cop feta la projecció per escollir els que seran bons mostràrem les contribucions al límit de Q de cada una de les observacions projectades. Si

estan dins els límits la afegiríem al model si no ho esta no li afegiríem.

Pas 1 - Creació del Model:

Crearem un nou objecte tal com hem explicat en aquesta mateixa secció. Un cop creat fixarem el mètode de treball en el nostre model. A la Figura17 podem observar un exemple de com inicialitzar l'objecte. Com que volem estudiar tot el lot al mateix temps hem de crear un model PCA en BW. Hem de recordar no utilitzar VRE en BW. Per la resta de paràmetres podem escollir la configuració que més ens agradi.

```
%Creem El PCA model
model=PCA(m);
model.setconv(0);%1=VW,0=BW
model.setEstandaritzacio(3);%1=AS,2=GS,3=CS,4=BS
%0=Manual,1=Kaiser,2=colze,3=colze+punt,4=%lambda,5=VRA
model.setMetode(1);
model.setMetodeLimits(1);%1=analitic,0=3sigma
model.setTreuFM(0);%1=treure,0=no treure
```

Figura 17: Codi Matlab®creem el model PCA

A la Figura17 es crea un model PCA. El model que creem treballara en BW, fara Continous Scaling a les dades, el mètode de selecció de components principals serà pel criteri de Kàiser (totes les majors de 1), els límits es calcularan analíticament i no traurem la forma mitja de les dades.

Pas 2 - Creació de la projecció:

Com que ja tenim el model on projectar les dades, tot seguit caldrà crear un objecte Dades per guardar el conjunt de proves (el que mes endavant projectarem al model). Per saber com crear un objecte Dades podem fixar-nos amb qualsevol dels exemples de l'apartat utilització de Dades. Un cop creat un objecte dades amb el conjunt de proves procedim a projectar-lo al model, a la Figura18 tenim la comanda que ens ho fa.

```
%creem un nou PCA projectant dades al model
projectat=model.projecta(p);
```

Figura 18: Codi Matlab®creem el model PCA projectat

Un cop creats els 2 objectes PCA podem utilitzar la comanda "plotT2Q" per fer-nos una idea de com estan situats els diferents casos. A la Figura19 hi tenim la representació en l'espai T2 contra Q de tots els casos utilitzats per fer el model, a la Figura20 hi tenim la dels elements a comprovar. Amb les nostres dades podem observar que tots els casos del model son dins els límits de T2 i Q, en canvi dels elements a comprovar no ni ha cap.

Pas 3 - Preguntar que fer a l'usuari:

Aquí caldrà mostrar les contribucions al límit de Q de cada lot. Ho farem una a un i es mostraran successivament un rera l'altre mentre demanem a l'usuari que cal fer amb el lot que esta veient per pantalla. Amb la funció

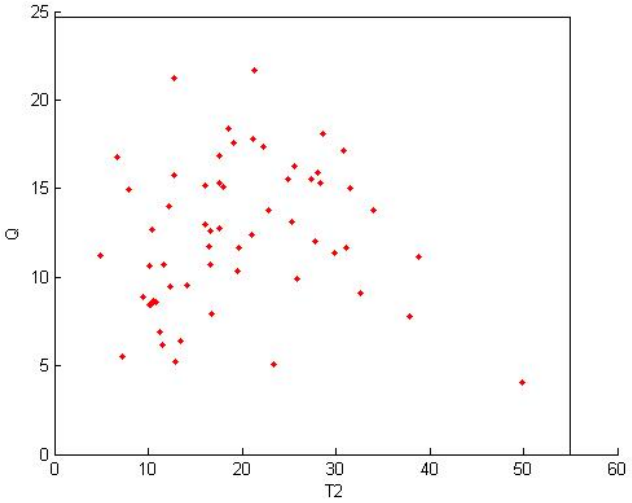


Figura 19: model.plotT2Q

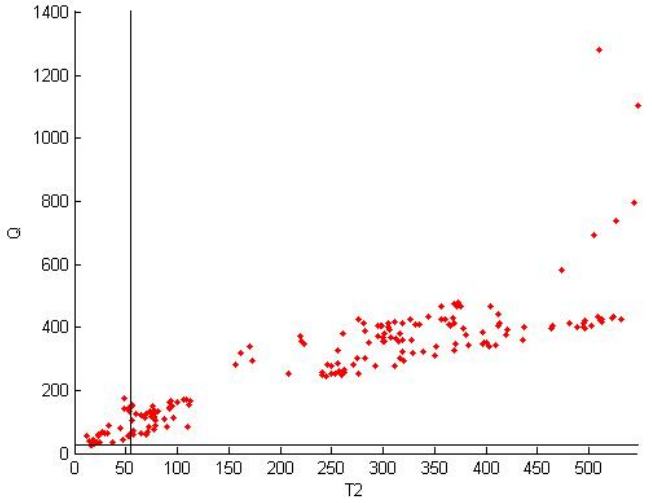


Figura 20: projectat.plotT2Q

"plotContLimQ" sens simplificara molt la feina, només cal indicar-li quin lot mostrar. a la Figura21 hi tenim el tros de codi corresponent.

```
for i=250:p.getnlots()
    projectat.plotContLimQ(i)
    %demanem que fer a l'usuari
    resposta=input('S"ha d"afegir?( 1=si 0=no)');
    %afegim la resposta a on toca
    e(i)=(resposta==1);
end
```

Figura 21: Codi Matlab®creem el model PCA projectat

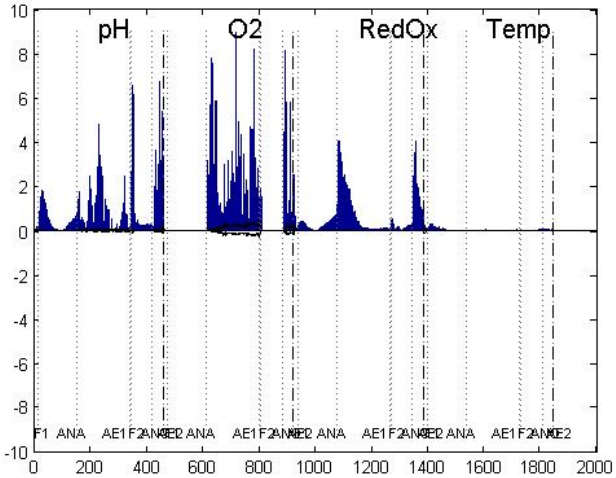


Figura 22: projectat.plotContLimQ(1)

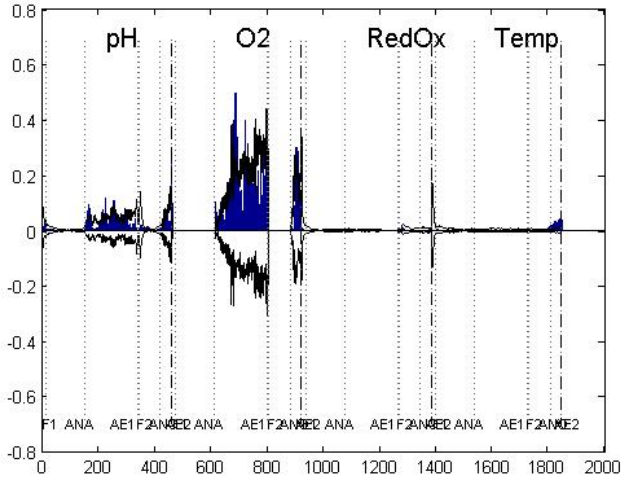


Figura 23: projectat.plotContLimQ(250)

Les Figures22 i 23 mostren el resultat de la crida "projectat.plotContLimQ(1)" i "projectat.plotContLimQ(250)" respectivament. Veiem que el primer cas es clarament dolent, en canvi el segon correspon a una observació molt més propera als límits.

Pas 4 - Afegir al model els que ha triat l'usuari:

Per afegir els lots seleccionats nomes cal passar el vector de logicals que hem anat creant a "getlot". Aquesta funció ens retornara els lots i les classes seleccionades, El retorn el passarem directament per paràmetre a la funció "afegeixlot" que ens els afegira al model. Com podem veure a la Figura24 la funció "afegeixlot" ens borrara les dades que s'havien anat calculant internament (dades estandaritzades, limits, Components principals...) perquè

em modificat les dades i ja no són correctes. Quan es tornin a necessitar es recalcularan.

```
if sum(e)>0 %comprobem que no sigui buit
    [data ,class] =projectat.getlot(e);
    model.afegeixlot(data,class);
end
```

Figura 24: Codi Matlab®creem el model PCA projectat

Part IV

CBR

12 Descripció

Aquesta Classe utilitza les dos anteriors (PCA i Dades). L'objectiu de la classe CBR és tal com indica el seu nom aplicar tècniques de raonament basat en casos (Case-based reasoning). Aquesta classe permetrà l'aprenentatge a partir d'exemples i la posterior recuperació i reutilització d'aquests per resoldre casos similars.

13 UML

La Figura25 mostra la representació utilitzant la notació UML de la classe CBR. Les files en vermell indiquen les funcions i atributs privats. Les funcions marcades amb blau són els constructors de a classe. Finalment les funcions marcades amb color verd són els mètodes amb accés públic visible.
La seva funcionalitat principal serà crear i gestionar la nostra base de casos. Afegint o eliminant els casos que faci falta per mantenir la base de casos més adient en tot moment.

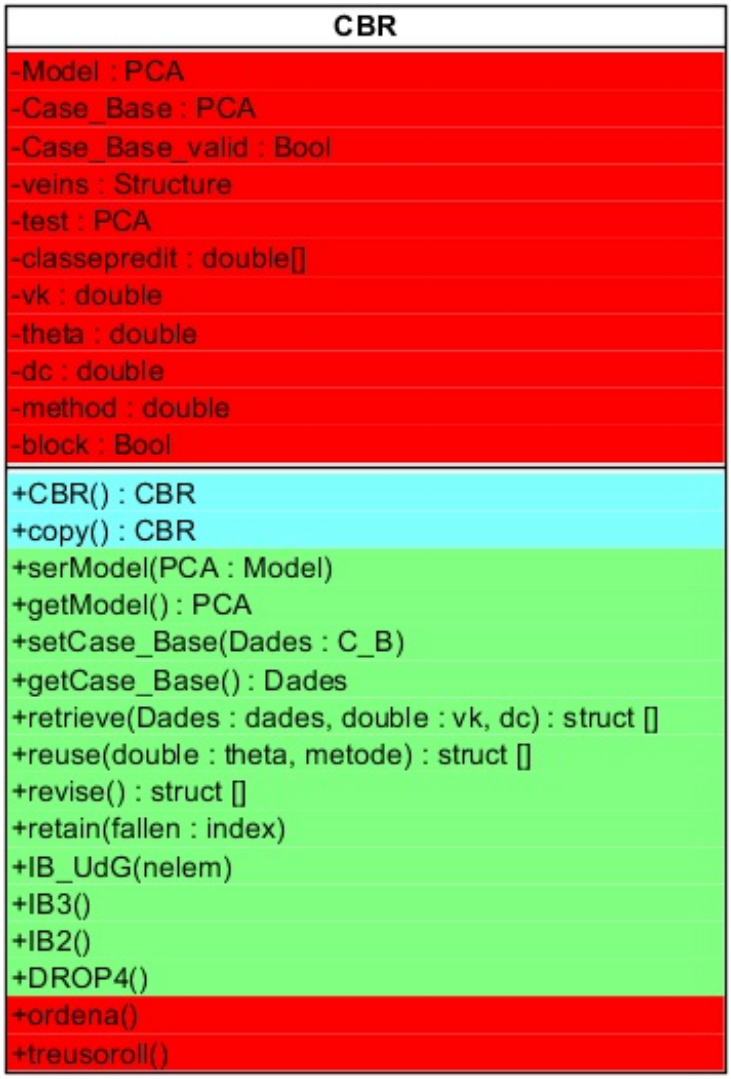


Figura 25: Classe Dades

14 Utilització

CBR depèn molt de PCA, perquè l'utilitzarem per crear el model del problema i la nostra base de casos serà un PCA "projectat" al model. Amb Dades també hi tenim una dependència clara, no nomes perquè en necessitarem una instància per crear els objectes PCA si no que en alguns mètodes els resultats poden variar segons l'ordre dels lots.

El primer que hem de fer per treballar amb la classe CBR és crear-ne una instància. Per fer-ho podem copiar-ne una de ja existent invocant el seu mètode "copy()" o bé podem crear-ne una de nova amb el constructor "CBR()". A la Figura26 tenim un exemple de com utilitzar els constructors de la classe. Concretament el que fem és crear una instància (amb tots els camps buits per defecte) amb el constructor "CBR()" i després en fem una copia amb "copy()".

```
MY_CBR=CBR(); %crea una instancia buida
CBR_copia=MY_CBR.copy(); %crea una instancia copia
```

Figura 26: Codi Matlab®creem diferents objectes CBR

Un cop creada la nostra instància de CBR haurem de inicialitzar els atributs utilitzant els setters. Per poder començar a utilitzar el nostre objecte fixarem un model i una base de casos inicials amb "setModel()" i "setCase_Base()" respectivament.

Si analitzem pas per pas aquest procés veurem que el que cal fer és primer crear un objecte PCA que serà el model del nostre CBR a la Figura27 creem un model utilitzant totes les dades bones que tenim. En concret aquestes dades tenen 4 variables,424 observacions en el temps i disposem de 70 lots. Com a classe els hi assignem 1 que a partir d'ara serà per els bons.

```
%carreguem les dades
load Matriu_LEQUIA_Bons_2A_Final.mat;
load Matriu2A_Bons_3e.mat;

dat=Dades();%creem un nou objecte Dades
dat.setdataBW(Matriu2ABons,4,424,70);%Entrem les dades en el format correcte
dat.setvariables(Matriu.Vars);%necessitem que siguin algun non
cla1=ones(1,70);%creem el vector de classes, tot a 1 perquè son totes bones
dat.setclass(cla1);%afegim les classes a les dades del PCA

%Creem el Model PCA
M=PCA(dat);
```

Figura 27: Codi Matlab®on creem un model PCA pel nostre CBR

Seguidament crearem la base de casos. Com es pot observar a la Figura28 en el nostre cas posarem totes les dades de que disposem en un mateix objecte Dades (Que utilitzarem per base de casos), assignant 1 a les observacions bones, 2 a les observacions dolentes i 3 a les regulars.

```
%carreguem les dades
load Matriu2A_Dolents.mat;

cla2=ones(1,91)*2; %creem les classes posem tot a 2 son tots dolents
aux=Dades();
 aux.setdataBW(Matriu2ADolents,4,424,91); %Entrem les dades en el format correcte

%ajuntem Bons i dolents
dade=cat(3,aux.getdata3D(), dat.getdata3D());
claa=cat(2,cla2,cla1);

load 'Matriu_LEQUIA_Regulars_2A.mat';
dadess = Matriu.Variables;
cla3=ones(1,62)*3; %creem la classe 3 per ela regulars

%Ajuntem tambe els regulars
dade=cat(3,dade, Matriu.Variables);
claa=cat(2,claa,cla3);

dat1=Dades();%creem un objecte dades
dat1.setdata(dade);%li afegim les dades
dat1.setvariables(Matriu.Vars); %posem noms a les variables
dat1.setclass(claa); %afegim les classes
```

Figura 28: Codi Matlab®on creem una base de dades pel nostre CBR

Finalment un cop creats “M”(model) i “dat1”(base de casos) nomes caldrà afegir-los al nostre objecte CBR com fem a la Figura29.

```
MY_CBR.setModel(M);
MY_CBR.setCase_Base(dat1);
```

Figura 29: Codi Matlab®on afegim el model i la base de casos al nostre CBR

Arribats a aquest punt ja podem fer retrieve, reuse, revise , retain o optimitzar la base de casos amb metodes com Drop4, IB3...

Per tal d'invocar els mètodes d'optimització de la base de casos o retrieve i reuse disposem de 2 maneres distintes, la completa i la abreviada. A la Figura30 podem veure les crides completes, en contraposició a la Figura31 tenim les mateixes crides en forma abreviada. On “prov” és un objecte Dades, “nelem” i “descartats” 2 doubles que indiquen quants veïns agafem i quants en descartem.

La forma abreviada només la podrem utilitzar si els paràmetres (vk, dc, theta i method) s'han utilitzat en alguna crida anterior, l'objecte suposa que si no se li han passat són els mateixos.

```
vk = [5 3];
dc = 10;
theta = 0.3;
methode = 3;
MY_CBR.retrieve(prov,vk,dc);
MY_CBR.reuse(theta,method);
MY_CBR.IB2(vk,dc,theta,methode);
MY_CBR.IB3(vk,dc,theta,methode);
MY_CBR.IB_UdG(nelem,descartats,vk,dc,theta,methode);
MY_CBR.DROP4(vk,dc,theta,methode);
```

Figura 30: Codi Matlab®exemples de crides CBR 1

```
MY_CBR.retrieve(prov);
MY_CBR.reuse();
MY_CBR.IB2();
MY_CBR.IB3();
MY_CBR.IB_UdG(nelem,descartats);
MY_CBR.DROP4();
```

Figura 31: Codi Matlab®exemples de crides CBR 2

Part V

Apèndix I - Dades

A Atributs

Els atributs de la classe Dades són privats. El fet de tenir els atributs privats ens permet controlar el tipus i la coherència de les dades que ens hi entra l'usuari i així evitar problemes.

string : nom

Aquest camp és on l'usuari pot guardar el nom del procés, no és obligatori que un procés tingui nom, per tant aquest camp serà opcional. Aquest camp serà un string. Aquest atribut s'inicialitzarà a null només crear l'objecte i s'hi mantindrà fins que l'usuari el modifiqui amb el setter corresponent.

handle : info

Aquest camp esta pensat com a espai on l'usuari podrà guardar informació general sobre el procés des de dia de captura fins a operari que l'ha fet... En aquest camp l'usuari pot emmagatzemar qualsevol tipus de dades vàlid en Matlab®sense que això afecti al funcionament de l'objecte. Aquest atribut és totalment opcional i mentre l'usuari no hi hagi entrat alguna cosa es mantindrà sempre a null.

string[] : variables

En aquest camp s'han de guardar els noms de les diferents variables que intervinguin en el procés. Per facilitar el seu maneig aquest camp serà un vector de cel·les que contenen text "el nom de les variables". Aquesta informació serà important sobretot quan utilitzem les dades per fer PCA, si no tenim aquesta informació algunes de les funcionalitats de l'objecte PCA creat amb aquestes Dades podrien no funcionar correctament. Aquest camp és necessari i a més és important que el seu tamany sigui coherent amb el de les dades. El nombre de variables (la longitud del vector) ha de ser la mateixa que el nombre d'elements de la segona dimensió de l'atribut data.

string[] : fases

En aquest camp és on es guarden els noms de les diferents fases del procés. Per facilitar el seu maneig aquest camp serà un vector de cel·les que contindran el text. Aquest vector ha de tenir el mateix nombre d'elements que el de durafases. Aquest atribut s'inicialitzarà a null en crear l'objecte i s'hi mantindrà mentre l'usuari no hi entri les dades corresponents.

double[] : durafases

Aquest camp és un vector de doubles i ens indicarà la duració de cada una de les fases. Lògicament la duració de la primera fase correspondrà a durafases[1], la de la segona durafases[2] i així successivament. Aquest

vector ha de tenir el mateix nombre d'elements que el vector de fases. Aquest atribut s'inicialitzarà a null en crear l'objecte i s'hi mantindrà mentre l'usuari no hi entri les dades corresponents.

double[] : lots

Aquí es guardarà el nombre de lots que té el procés. Aquest paràmetre s'actualitza de forma automàtica en el moment d'entrar les dades (batch-wise, variable-wise o matriu 3D), així com quan s'utilitzin mètodes que modifiquen el nombre d'aquests elements (ajunta, afegeix, treu).Aquest atribut s'inicialitzarà a 0 en crear l'objecte i s'hi mantindrà mentre no tinguem dades a l'objecte. En cas d'eliminar tots els lots de l'objecte, aquest valor automàticament tornarà a ser 0.

string[] : reflots

Aquest camp és on guardarem el nom o referència de cada un dels lots. Aquest paràmetre es opcional, si treballem amb la classe CBR ens serà molt útil per saber la referència inicial dels lots que queden a la base de dades després d'optimitzar-la. Aquesta llista sofrira les mateixes modificacions que les Dades per les funcions Shuffle, ajunta, afegeix, de manera que el nom sempre estarà a la mateixa posició que el lot a qui pertany.

double[][][] : data

El camp data és on guardarem les dades del procés (internament és una

matriu de doubles 3D) i quan calgui en farem el desdoblament a BW o VW segons convingui. Les dimensions seran $dim1 = Temps$,$dim2 = Variables$ i $dim3 = Lots$. A la Figura32 podem observar una representació esquemàtica d'aquest atribut. Aquest atribut s'inicialitzarà a null en crear l'objecte i si mantindrà mentre l'usuari no hi entri les dades del procés.

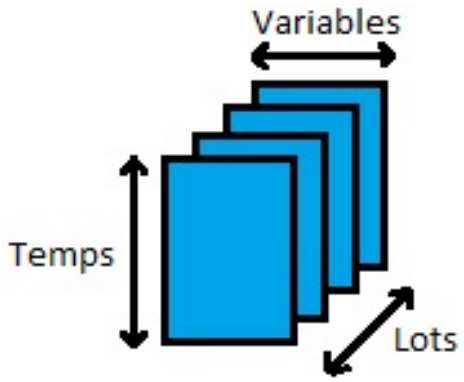


Figura 32: Dades en memòria

double[] : class

És un vector de doubles on es guarda la classe de cada batch en format numèric. És molt important, per assegurar un bon comportament de l'aplicació, que aquest camp tingui tants elements com lots hi hagi a les dades. Aquest atribut s'inicialitzarà a null en crear l'objecte i s'hi mantindrà mentre l'usuari no hi entri les classes dels lots.

B Funcions

B.1 Funcions Públiques

Tot seguit explicarem que fa cada una de les funcions a les que te accés l'usuari de la classe Dades.

B.1.1 Getters

Aquestes funcions són per definició les que ens permetran extreure informació de la nostra instància de l'objecte Dades.

getnom() : string

Aquesta funció és l'encarregada de recuperar el nom del procés que el nostre objecte dades guarda al seu atribut nom. Recordem que aquest camp no és obligatori, per tant pot ser que l'usuari no l'hagi entrat mai i contingui un valor per defecte(null).

Paràmetres de sortida:

string	És el nom del procés emmagatzemat a l'objecte

getinfo() : handle

Aquesta funció ens retornara el que hagi entrat l'usuari com a informació general del procés, Aquest camp no és obligatori utilitzar-lo, i pot prendre el seu valor per defecte (null).

Paràmetres de sortida:

handle — Aquest camp és de tipus handle, en Matlab®qualsevol tipus de dades vàlid és una subclasse de handle.

getnlots() : double

Aquesta funció es la que ens retorna el nombre de lots que formen les nostres dades. Els lots s'actualitzen sols i en cas d'efectuar una crida a aquesta funció quan no hi ha dades a l'objecte retornarà 0.

Paràmetres de sortida:

double — És el nombre de lots que tenim en aquest moment en les dades de l'objecte.

getnvars() : double

Aquesta funció es la que ens retorna el número de variables que formen les nostres dades. Aquesta funció quan no hi ha dades a l'objecte retornara 0.

Paràmetres de sortida:

double — És el número de variables que tenim en aquest moment en les dades de l'objecte.

getntemps() : double

Aquesta funció es la que ens retorna el número de instants de temps que formen les nostres dades. En cas d'efectuar una crida a aquesta funció quan no hi ha dades a l'objecte retornara 0.

Paràmetres de sortida:

double	És el número de instants de temps que tenim en aquest moment en les dades de l'objecte.

getvariables() : string[]

Retorna els noms de les diferents variables del procés. Les variables recordem que corresponen a la dimensió 2 de les dades emmagatzemades a l'objecte, per tant, és d'esperar que sigui un vector de tantes posicions com variables tinguin les dades. en cas de no estar inicialitzat ens trobarem que el seu valor serà null.

Paràmetres de sortida:

string[]	Ens retorna els diferents noms de les variables del procés

getdata3D() : double[][][]

Retorna les dades del procés en format de matriu 3D. Les retorna en format de matriu de doubles de 3 dimensions on cada dimensió correspondrà a $dim1 = Temps$,$dim2 = Variables$ y $dim3 = Lots$. Si les dades no han estat mai inicialitzades o s'han eliminat totes anteriorment a fer la crida aquesta funció retornara null.

Paràmetres de sortida:

double[][][]	Les dades en format [temps][variables][Lots]. A la Figura33 podem veure una representació del formar d'aquest valor de retorn.

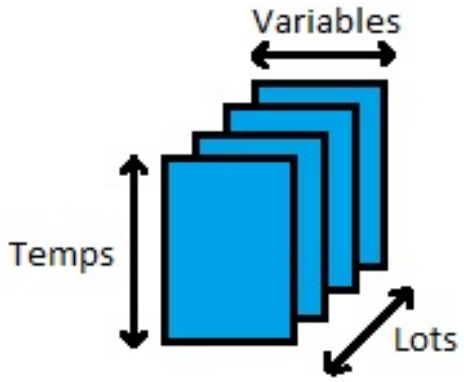


Figura 33: Dades

getfases() : string[]

Retorna els noms de les diferents fases del procés. Aquest valor pot no estar inicialitzat, si és el cas retornara null.

Paràmetres de sortida:

string[]	Són els noms de les diferents fases que te el nostre procés

getdurafases() : double[]

Retorna la duració de les fases del procés. Aquest valor pot no estar inicialitzat, si és el cas retornara null.

Paràmetres de sortida:

double[]	Són les duracions de les diferents fases que te el nostre procés

getclass() : double[]

Retorna la classe dels diferents lots del procés. Les classes seran sempre

doubles. Aquesta funció retornara aquests doubles corresponents a les classes de cada lot. L'atribut class pot no estar inicialitzat o bé que s'hagin tret totes les dades, si això passa retornara null.

Paràmetres de sortida:

double[]	És un vector de doubles i corresponenen a les classes de cada lot.

getlot(index i) : struct[double[][][] , double]

Retorna el lot o grup de lots de dades "indexats" per "i" i la seva classe. En cas d'intentar aconseguir un lot inexistent aquesta funció donara error i parara l'execució.

Paràmetres d'entrada:

i	Pot ser un double, un vector de doubles, un vector de logicals o qualsevol altre tipus vàlid en indexacións en Matlab®.

Paràmetres de sortida:

double[][][]	Són les dades dels diferents lots demanats al paràmetre i.
double[]	Són les diferents classes dels lots que hem demanat al paràmetre i.

getnomlots() : string[]

Retorna un vector amb els noms de tots lots del procés en el mateix ordre que es troben a les dades. Els noms dels lots de les Dades poden no estar inicialitzades o bé que s'hagin tret tots les lots, si això passa retornara null.

Paràmetres de sortida:

string[]	És un vector de noms, corresponents a cada lot.

B.1.2 Setters

Aquestes funcions són per definició les que ens permetran introduir informació a la nostra instància de l'objecte Dades.

setnom(string nom) : Void

Funció per entrar el nom a l'objecte actual. El paràmetre nom ha de ser una cadena de caràcters, si no el mètode mostrara un error i es finalitzara l'execució.

Paràmetres d'entrada:

nom	És un string i serà el nou nom del procés.

setinfo(handle info) : Void

És una funció per donar la informació que l'usuari necessiti sobre el procés. Aquest camp és totalment lliure, com que el Matlab®ens ho permet l'usuari podrà entrar qualsevol tipus de dades vàlid que ell cregui necessari guardar sobre el procés dins l'objecte dades.

Paràmetres d'entrada:

info	És una variable on l'usuari pot entrar-hi qualsevol cosa que cregui convenient mentre sigui un tipus valid de dades pel Matlab®.

setvariables(string[] variables) : Void

Aquesta funció fixarà els noms de les variables. Han de seguir el mateix ordre que les variables enregistrades, i nomes acceptem un nom per cada variable.

Paràmetres d'entrada:

variables	És el nou vector de noms de les variables. Aquest vector ha de ser una llista de cel·les que continguin un text (el nom de la variable)

setdata(double[][][] data) : Void

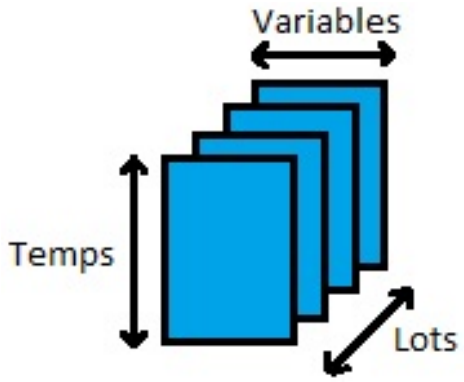
Carrega la matriu de dades, aquest mètode carrega les dades directament a l'atribut dades del nostre objecte, cal vigilar molt el format que tenen les dades si no pot portar problemes.

Format correcte:

dim 1 =Temps

dim 2 =Variables

dim 3 =Lots



Paràmetres d'entrada:

data	Es una matriu de doubles amb el format descrit anteriorment. Aquesta matriu seran les noves dades de l'objecte actual.

setdataBW(double[][] data, double vars, double temp, double lots) : Void

Carrega la matriu de dades. S'espera una matriu desdoblada en bach_wise, per tant ha de ser una matriu 2D i el format esperat és:

dim 1 =Lots

dim 2 =Variables*Temps

Com podem veure a la representació de la Figura2

Paràmetres d'entrada:

data	Ha de ser una matriu 2D de doubles, són les dades que bolem posar al nostre objecte.
vars	És el nombre de variables del procés també com a double.
temp	És el nombre de mostres al llarg del temps del nostre procés.
lots	És numero de lots que ens han entrat.

setdataVW(double[][] data, double vars, double temp, double lots) : Void

Carrega la matriu de dades. S'espera una matriu 2D en format variable-wise. Les dades les esperem amb el següent format:

dim 1 =Lots*Temps

dim 2 =Variables

Com podem veure a la representació de la Figura3

Paràmetres d'entrada:

data	Ha de ser una matriu 2D de doubles, són les dades que bolem posar al nostre objecte.
vars	És el nombre de variables del procés també com a double.
temp	És el nombre de mostres al llarg del temps del nostre procés.
lots	És numero de lots que ens han entrat.

setfases(string[] data) : Void

Utilitzant aquesta funció podrem entrar els noms de les fases del procés.

Paràmetres d'entrada:

data	Seran els noms de les fases del procés, ha de ser un vector de cel·les amb text.

setdurafases(double[] data) : Void

Amb aquesta funció podrem entrar les que seran les noves duracions de les fases del procés, han de ser un vector de doubles de la mateixa longitud que el nombre de fases que te el nostre procés.

Paràmetres d'entrada:

data	És un vector de doubles amb la duració de cada una de les fases

setclass(double[] data) : Void

Funció per fixar les classes de cada lot de dades. Les classes hauran de ser doubles.

Paràmetres d'entrada:

data — Seran les noves classes del nostre procés actual, han de ser un vector de doubles de la mateixa longitud que el nombre de fases del procés. O també pot ser un vector de cells amb text.

setnomlots(string[] data) : Void

Funció per fixar el nom de cada lot de dades. El nom de cada lot pot ser el que desitgi l'usuari. El nom pot utilitzar-se com a identificador del lot.

Paràmetres d'entrada:

data — Seran els noms dels lots. El vector ha de tenir un nom per cada lot es a dir ha de tenir longitud = al nombre de lots.

B.1.3 Altres

Aquestes son les funcions especifiques per l'objecte Dades, aquí trobarem funcions per afegir lots, treure lots, ajuntar 2 objectes de tipus dades,constructor de la classe...

ajunta(Dades dad) : Dades

Aquesta funció ajuntara 2 objectes dades i retornara un 3r objecte format per la unió dels 2, si cridem "x.ajunta(Dades)" llavors "x" serà l'objecte principal, d'ell s'agafara el nom, temps, duració de fases... de l'objecte Dades dad nomes utilitzarem les dades i les classes. Podem veure visualment el que fa aquesta funció a la Figura34.

dim 1 =Temps

dim 2 =Variables

dim 3 =Batch

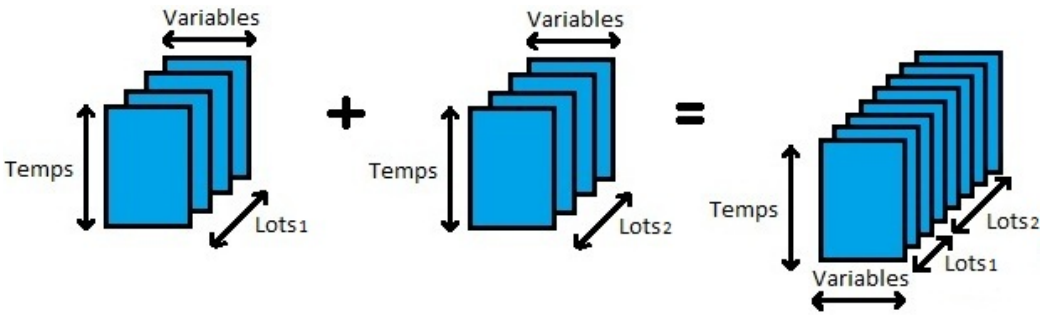


Figura 34: Ajunta

Paràmetres d'entrada:

dad	Ha de ser un objecte Dades, serà l'objecte secundari el primari es l'actual.

Paràmetres de sortida:

Dades	Es el nou objecte resultat de ajuntar l'actual amb dad

afegeixlot(double[][][] dad, double[] clase) : Void

Aquesta funció afegeix un lot o un grup de lots a les nostres dades amb

les corresponents classes. Les mides del nou lot o grup de lots han de concordar amb les de l'atribut dades del nostre objecte.

Paràmetres d'entrada:

dad	Pot ser un lot de dades o uns quants lots de dades.
classe	Són les classes associades a cada un dels lots de dad.

Paràmetres de sortida:

Void

treulot(index i) : Void

Aquesta funció treurà el lot o lots indexats per "i". Si l'usuari intenta treure un lot no existent aquesta funció mostrara un error i finalitzara l'execució.

Paràmetres d'entrada:

i	Indica el grup de lots que volem treure de les nostres dades

Paràmetres de sortida:

Void

Dades() : Dades

És el constructor per defecte: Aquest és el mètode constructor de l'objecte Dades. Per defecte es fixen tots els valors a null esperant que l'usuari entri les dades que realment hi han d'anar, podríem dir que nomes reservem l'espai per l'objecte. Ells Lots son l'únic atribut que

s'inicialitzarà a 0, ja que si no hi ha dades tenim 0 lots.

Paràmetres d'entrada:

Void

Paràmetres de sortida:

Dades	Un nou objecte de tipus Dades

copy() : Dades

Aquest constructor farà una copia del nostre objecte i la retorna per referència. No es diu com la classe ja que el Matlab®no permet el sobrecarregar mètodes d'un objecte.

Paràmetres d'entrada:

Void

Paràmetres de sortida:

Dades	Retorna un nou objecte Dades, copia de l'actual.

referencia() : Void

És un mètode molt simple però necessari, la seva feina es retornar una referència de l'objecte amb tot el que això implica.

Paràmetres d'entrada:

Void

Paràmetres de sortida:

Dades	Retorna una nova referència a l'objecte l'actual.

suffle() : Void

És un mètode que reordena aleatòriament els lots del nostre objecte Dades.Aquesta funció serà útil per quan bolguem utilitzar IB2, IB3 o DROP4 quan treballem amb un objecte CBR.

Paràmetres d'entrada:

Void

Paràmetres de sortida:

Void

save(string fitxer,string format) : Void

Aquest mètode guarda l'objecte Dades actual en un fitxer. El fitxer en cas d'existir serà sobrescrit, altrament el creara de nou. El paràmetre fitxer ha de contenir el nom que vulguem donar al fitxer i si cal la ruta absoluta d'on l'hem de crear. El paràmetre format ens indicara el format de les Dades. Actualment tenim definits 2 formats 'mat' i 'csv'. El format 'mat' guardara l'objecte amb el format per defecte de Matlab®, en canvi l'opció 'csv' creara un fitxer en un format propi pensat per facilitar l'intercanvi d'informació amb altres aplicacions. Per més informació sobre el format 'csv' veure annex CSV.

Paràmetres d'entrada:

fitxer	Aquest paràmetre ha de contenir el nom i extensió (en cas que se'n vulgui) del fitxer on vulguem guardar.
format	Aquest paràmetre indica el format que tindrà el fitxer.

Paràmetres de sortida:

Void

load(string fitxer,string format) : Void

Aquest mètode llegirà un fitxer i inicialitzarà l'objecte Dades actual amb la informació del fitxer. El paràmetre fitxer ha de contenir el nom i si cal la ruta absoluta d'on hi ha el fitxer. El paràmetre format ens indicara el format en que es troben les dades del fitxer. Actualment tenim definits 2 formats 'mat' i 'csv'. El format 'mat' guardara l'objecte amb el format per defecte de Matlab®, en canvi l'opció 'csv' indica que el fitxer ha estat creat amb el nostre propi format pensat per facilitar l'intercanvi d'informació amb altres aplicacions. Per més informació sobre el format 'csv' veure annex CSV.

Paràmetres d'entrada:

fitxer	Aquest paràmetre ha de contenir el nom i extensió (en cas que se'n vulgui) del fitxer on vulguem guardar.
format	Aquest paràmetre indica el format que tindrà el fitxer.

Paràmetres de sortida:

Void

B.2 Funcions Privades

En aquest apartat explicarem que fa cada una de les funcions ocultes a l'usuari de la classe Dades.

to2D(double metode) : double][][]

Mètode privat de la classe Dades que converteix la matriu de dades interna a 2D, si mètode es 1 fara un unfolding per variable-wise si es 0 el fara per bach_wise.

Paràmetres d'entrada:

metode	Ens indica el mètode d'unfolding a utilitzar

Paràmetres de sortida:

double[][]	Són les dades resultants de l'unfolding

Part VI

Apèndix II - PCA

C Atributs

Com en el cas de l'objecte Dades aquí els atributs de la classe també són privats.

Les dades més importants que emmagatzemarem en aquest objecte són:

Dades : Dades

Serà l'objecte tipus Dades on hi haurà les nostres dades inicials en tot moment. Per poder utilitzar PCA és obligatori tenir un objecte Dades. Si no passem un objecte Dades al constructor PCA no ens deixara crear una nova instància de PCA.

bool :estàndard

És la variable que ens indica si el procés esta estandaritzat o no ho esta. Internament ens servira per saber si cal estandaritzar les dades o no abans de executar pca. Per defecte s'inicialitzarà a no estandaritzat = 0.

bool : model

Aquesta variable ens indica si les dades emmagatzemades a l'objecte

formen un model o provenen d'una projecció sobre un model extern. Depenent de l'origen de l'objecte ens trobem que tenim PCA-model i PCA-projeccions. Els models es creen amb el constructor de la classe. Les projeccions es creen amb el mètode projecta. Les projeccions tenen algunes característiques especials, com per exemple que la mitja, desviació estàndard i altres atributs els venen prefixats pel seu model.

double[][] : Z

Aquí es on guardarem una matriu amb les dades del procés estandaritzades.

double[][] : P

Aquí guardarem una matriu amb les components principals del procés.

double : Num

És el numero de components principals que s'utilitzaran per fer el model.

double[][] : T

És la matriu de T resultant d'aplicar l'anàlisi de components principals.

double[][] : Lambda

Aquí guardarem la matriu de lambdas.

double[][] : Lambdai

Aquí guardarem la matriu de lambdas inicials.

double[] : sta

És la desviació estàndard de les dades del model.

double[] : mea

És la mitjana de les dades del model.

double[][] : E

L'Error de cada element del model.

double[] : T2

És la matriu T^2.

double[] : LimT2

Límit de T^2.

double[] : LimQ

Límit de Q.

double : metode

És el mètode de selecció del nombre de components principals que s'utilitzarà. Els possibles valors són:

0 = Manual - Demanem a l'usuari per consola que ens entri el numero de components que vol utilitzar.

1 = Kaiser (opció per defecte) - Descarta totes les components que reconstrueixen menys de una variable inicial.

2 = Colze - Mostrem la gràfica de les lambdas i demanem a l'usuari que entri el punt on hi ha el colze.

3 = Colze + punt recomanat - Com el mètode anterior però recomanem un punt.

4 = % de lambda explicada - Demanem a l'usuari el % de variància que vol que quedi explicada al model i escollim el nombre de components principals perquè es compleixi.

5 = VRE - Aplica el mètode extern de *(Variance Reconstruction Error)*

double : metodee

És el mètode d'estandardització que s'utilitzara i els valors són:

1=autoscaling (opció per defecte)

2=Grup Scaling

3=Continous Scaling

4=Block Scaling

bool : treuFM

Indica si volem treure la forma mitja de les dades 1=si(opció per defecte) i 0=no nomes afecta si estem treballant en VW

bool : conv

- Fixara el mètode de conversió on:

0 = BW (opció per defecte)

1 = VW

VRE_Struct : struct

Estructura de dades on guardarem tot el que necessitin els mètodes externs per fer VRE.

double[] : CLimQstd

Límit de Q - Desviació típica.

double[] : CLimQmea

Límit de Q - Mitjana.

double[] : CLimT2std

Límit de T2 - Desviació típica.

double[] : CLimT2mea

Límit de T2 - Mitjana.

D Funcions

D.1 Funcions Publiques

Tot seguit explicarem que fa cada una de les funcions a les que te accés l'usuari de la classe PCA.

D.1.1 Getters

Aquestes funcions són per definició les que ens permetran extreure informació de la nostra instància de l'objecte PCA.

getdata() : double[][]

Retorna les dades originals. Sempre que no s'hagi modificat l'objecte Dades. Les retornara en BW o VW segons el mètode que estiguem treballant amb l'objecte actual.

Paràmetres de sortida:

double[][]	És una matriu 2D en format BW o VW segons s'hagi especificat amb el mètode de conversió..

getDades() : Dades

Retorna una copia de l'objecte Dades de l'objecte actual.

Paràmetres de sortida:

Dades	Retorna una copia de l'objecte que tenim assignat al nostre PCA com a Dades

getlot(index i) : struct[double[][][] , double]

Retorna el lot o grup de lots de dades "indexats" per "i" i la seva classe. En cas d'intentar aconseguir un lot inexistent aquesta funció donara error i parara l'execució.

Paràmetres d'entrada:

i	Pot ser un double, un vector de doubles, un vector de logicals o qualsevol altre tipus vàlid en indexacións en Matlab®.

Paràmetres de sortida:

double[][][]	Són les dades dels diferents lots demanats al paràmetre i.
double[]	Són les diferents classes dels lots que hem demanat al paràmetre i.

getZ() : double[][]

Retorna les Dades estandaritzades.

Paràmetres de sortida:

double[][]	Són les dades estandaritzades

getP() : double[][]

Retorna La matriu de components principals.

Paràmetres de sortida:

double[][]	Retorna La matriu de components principals.

getT() : double[][]

Retorna La matriu de T.

Paràmetres de sortida:

double[][]	La matriu de T.

getLambda() : double[][]

Retorna La matriu de Lambdas.

Paràmetres de sortida:

double[][]	Retorna La matriu de Lambda.

getStd() : double[]

Retorna la desviació estàndard de les dades de la mostra.

Paràmetres de sortida:

double[]	Retorna la desviació estàndard de les dades.

getmean() : double[]

Retorna la mitjana de les dades de la mostra.

Paràmetres de sortida:

double[]	Retorna la mitjana de les dades.

getE() : double[][]

Retorna la matriu d'errors.

Paràmetres de sortida:

double[][]	Retorna la matriu d'errors.

getT2() : double[]

Retorna la matriu de T^2.

Paràmetres de sortida:

double[][]	Retorna la matriu de T^2.

getLimT2() : double[]

Retorna el límit de T^2.

Paràmetres de sortida:

double[]	Retorna el límit de T^2.

getLimQ() : double[]

Retorna el límit de Q.

Paràmetres de sortida:

double[]	Retorna el límit de Q.

getnum() : double

Retorna el numero de components principals utilitzades.

Paràmetres de sortida:

double	És el numero de components principals.

esmodel(): bool

Ens retorna Cert si l'objecte PCA conte un model o False si no ho és.

Paràmetres de sortida:

bool	Retorna 1 si el PCA actual és un model 0 si no ho és.

getMetode() : double

Retorna el numero de components que s'utilitza per escollir el numero de components. Els possibles valors són:

0 = Manual - Demanem a l'usuari per consola que ens entri el numero de components que vol utilitzar.

1 = Kaiser (opció per defecte) - Descarta totes les components que reconstrueixen menys de una variable inicial.

2 = Colze - Mostrem la gràfica de les lambdas i demanem a l'usuari que entri el punt on hi ha el colze.

3 = Colze + punt recomanat - Com el mètode anterior però recomanem un punt.

4 = % de lambda explicada - Demanem a l'usuari el % de variància que vol que quedi explicada al model i escollim el nombre de components principals perquè es compleixi.

5 =VRE - Aplica el metode de (Variance Reconstruction Error).

Paràmetres de sortida:

double	És el numero de mètode utilitzat.

getMetodeLimits() : double

Retorna el mètode que s'ha utilitzat per calcular els Limits.1=analitic i 0=3_sigma(per defecte).

Paràmetres de sortida:

double	És el mètode utilitzat. 1=analitic i 0=3_sigma(per defecte)

getEstandaritzacio() : double

Retorna el mètode d'estandardització utilitzat. Es mostra un "menu"per consola on es veuen els valors i significats possibles de l'atribut estandardització.

Paràmetres de sortida:

double	És el numero de mètode utilitzat.

getconv() : double

metode que retornar el metode de conversió que s'ha utilitzat 1= VW i 0=BW

Paràmetres de sortida:

double	És el mètode utilitzat.

getQ() : double[][]

Retorna la matriu de Q que tenim emmagatzemada dins el nostre objecte, si encara no la tenim la calculem.

Paràmetres de sortida:

double[][]	Retorna la matriu de Q.

getcontQ() : double[]

Aquesta funció retorna les contribucions de cada observació i variable a Q.

Paràmetres de sortida:

double[]	Són les contribucions de Q.

getcontT2() : double[]

Aquesta funció retorna les contribucions de cada observació i variable a T^2.

Paràmetres de sortida:

double[]	Són les contribucions de T^2.

getLimcontQmean() : double[]

Aquesta funció ens retornara la contribució a la mitjana del límit de Q per cada variable.

Paràmetres de sortida:

double[]	Són les contribucions a la mitjana del límit de Q per cada variable del nostre objecte PCA.

getLimcontQstd() : double[]

Aquesta funció ens retornara la contribució a la desviació estàndard del límit de Q per cada variable.

Paràmetres de sortida:

double[]	Són les contribucions a la desviació estàndard del límit de Q per cada variable del nostre objecte PCA.

getLimcontT2mean() : double[]

Aquesta funció ens retornara la contribució a la mitjana del límit de T^2 per cada variable.

Paràmetres de sortida:

double[]	Són les contribucions a la mitjana del límit de T^2 per cada variable del nostre objecte PCA.

getLimcontT2std() : double[]

Aquesta funció ens retornara la contribució a la desviació estàndard del límit de T^2 per cada variable.

Paràmetres de sortida:

double[]	Són les contribucions a la desviació estàndard del límit de T^2 per cada variable del nostre objecte PCA.

getclass() : double[]

Retorna la classe dels diferents lots del procés. Les classes seran sempre doubles. Aquesta funció retornara aquests doubles corresponents a les classes de cada lot. L'atribut class de Dades pot no estar inicialitzat o bé que s'hagin tret totes les dades, si això passa retornara null.

Paràmetres de sortida:

double[]	És un vector de doubles, corresponents a les classes de cada batch.

getnomlots() : string[]

Retorna un vector amb els noms de tots lots del procés en el mateix ordre que es troben a les dades. Els noms dels lots de les Dades poden no estar inicialitzades o bé que s'hagin tret tots les lots, si això passa retornara null.

Paràmetres de sortida:

string[]	És un vector de noms, corresponents a cada lot.

D.1.2 Setters

Aquestes funcions són per definició les que ens permetran introduir informació a la nostra instància de l'objecte PCA.

setMetode(double :x) : Void

Fixem el mètode que s'utilitzara per seleccionar el numero de components. Els possibles valors són:

0 = Manual - Demanem a l'usuari per consola que ens entri el numero de components que vol utilitzar.

1 = Kaiser (opció per defecte) - Descarta totes les components que reconstrueixen menys de una variable inicial.

2 = Colze - Mostrem la gràfica de les lambdas i demanem a l'usuari que entri el punt on hi ha el colze.

3 = Colze + punt recomanat - Com el mètode anterior però recomanem un punt.

4 = % de lambda explicada - Demanem a l'usuari el % de variància que vol que quedi explicada al model i escollim el nombre de components principals perquè es compleixi.

5 = VRE - Aplica el metode de (Variance Reconstruction Error).

Paràmetres d'entrada:

x	ha de ser de tipus double, serà el numero de mètode per escollir en nombre de components principals

setMetodeLimits(double :x) : Void

Fixem el mètode que s'utilitzara per calcular els límits. si es 1 = analitic 0 = 3_sigma (per defecte)

Paràmetres d'entrada:

x	Ha de ser de tipus double, serà el numero de mètode per calcular els límits.

setEstandaritzacio(double : x) : Void

Fixem el mètode d'estandardització que s'utilitzara.

Paràmetres d'entrada:

x	Ha de ser de tipus double, serà el nou mètode d'estandardització. Els valors possibles son: 1 - Auto Scaling 2 - Group Scaling 3 - Continous Scaling 4 - Block Skaling

setTreuFM(bool : x) : Void

fixem si es treurà la forma mitja de les dades en el cas d'estar treballant en VW, amb 1 es treurà amb la resta de valors no.

Paràmetres d'entrada:

x	Ha de ser de tipus double, s'espera un 1 o un 0, si és 1 forçaríem treure la forma mitja de les dades.

setconv(double : x) : Void

mètode per fixar el mètode de conversió que es vol utilitzar 1= VW i 0=BW.

Paràmetres d'entrada:

x	Ha de ser de tipus double, s'espera un 1 o un 0, si es 1 a partir d'ara treballarem en VW si es un 0 treballarem en BW.

setclass(double[] : class) : Void

Funció per fixar les classes de cada lot de les nostres dades. Les classes hauran de ser doubles.

Paràmetres d'entrada:

class	La nova classe de les nostres dades.

setnomlots(string[] data) : Void

Funció per fixar el nom de cada lot de dades. El nom de cada lot pot ser el que desitgi l'usuari. El nom pot utilitzar-se com a identificador del lot.

Paràmetres d'entrada:

data — Seran els noms dels lots. El vector ha de tenir un nom per cada lot es a dir ha de tenir longitud = al nombre de lots.

D.1.3 Altres

Aquestes són les funcions especifiques per l'objecte PCA, aquí trobarem funcions per afegir lots, treure lots, crear fitxers per entrar les dades al weka,constructor de la classe...

PCA(Dades : data):PCA

Es el constructor per defecte de l'objecte. Demana un objecte de tipus Dades per crear un objecte PCA, un PCA sense dades no te cap sentit. Carrega les Dades i Inicialitza tots els atributs de l'objecte a buits.

Paràmetres d'entrada:

Dades — Un objecte de tipus Dades per poder fer PCA

Paràmetres de sortida:

PCA — Un objecte de tipus PCA

copy():PCA

Constructor de copia, crea una copia del nostre objecte i el retorna per referència.

Paràmetres d'entrada:

Void

Paràmetres de sortida:

PCA	Un objecte de tipus PCA

weka() : file

Creara un fitxer anomenat Weka_dades.arff i hi posara les dades necessaris per entrar la base de casos al weka.

pca():[P T L]

Retorna les components principals.

Paràmetres d'entrada:

Void

Paràmetres de sortida:

P	La nova matriu de P del nostre PCA
T	La nova matriu de T del nostre PCA
L	La nova matriu de Lambdas del nostre PCA

projecta(Dades : x) : PCA

Projecta les dades sobre el nostre model i ens retorna un nou objecte.

Paràmetres d'entrada:

x	Les dades que volem projectar al nostre model

Paràmetres de sortida:

PCA	Un objecte de tipus PCA

reconstrueix() : struct[double[][], double, double, bool]

Si s'ha utilitzat VRE fa una reconstrucció per recuperar les dades originals.

Paràmetres d'entrada:

Void

Paràmetres de sortida:

double[][]	És la reconstrucció de les dades inicials.
double	La direcció de falla.
double	numero de vegades que s'ha cridar el mètode extern compute_VRE_sensor_reconstruction.
Bool	Cert si s'ha reconstruït be, fals si no s'ha pogut reconstruir

afegeixlot(double[][][] dad, double[] clase) : Void

Aquesta funció afegeix un lot o un grup de lots a les nostres dades amb les corresponents classes. Les mides del nou lot o grup de lots han de concordar amb les de l'atribut dades del nostre objecte. En cas de ser aplicada sobre un model caldrà recalcular totes les dades calculades fins al moment (afegir dades modifica la mitja). En canvi si s'aplica sobre un PCA no model afegirà les dades estandaritzant-les sense que afecti

a les dades precalculades. En cas d'haver de recalcular dades es fara automàticament a mesura que l'usuari les necessiti per evitar calcular coses innecessàries.

Paràmetres d'entrada:

dad	Pot ser un lot de dades o uns quants lots de dades.
classe	Són les classes associades a cada un dels lots de dad.

Paràmetres de sortida:

Void

treulot(index i) : Void

Aquesta funció treurà el lot o lots indexats per "i". Si l'usuari intenta treure un lot no existent aquesta funció mostrara un error i finalitzara l'execució. En cas de ser aplicada sobre un model caldrà recalcular totes les dades calculades fins al moment (treure dades modifica la mitja). En canvi si s'aplica sobre un PCA no model afegirà les dades estandaritzant-les sense que afecti a les dades precalculades. En cas d'haver de recalcular dades es fara automàticament a mesura que l'usuari les necessiti per evitar calcular coses innecessàries.

Paràmetres d'entrada:

i	Indica el grup de lots que volem treure de les nostres dades

Paràmetres de sortida:

Void

plotContLimT2(double[] : i) : Void

El paràmetre "i"és el batch o llista de batch que volem veure. Alguns exemples vàlids serien 1, [1 2 3 4], 1:5, : ...

El resultat serà un gràfic semblant al de la Figura35. Podem observar que es mostren les contribucions al límit de T2 i es subdivideix en Fases (part superior), variables (part inferior).

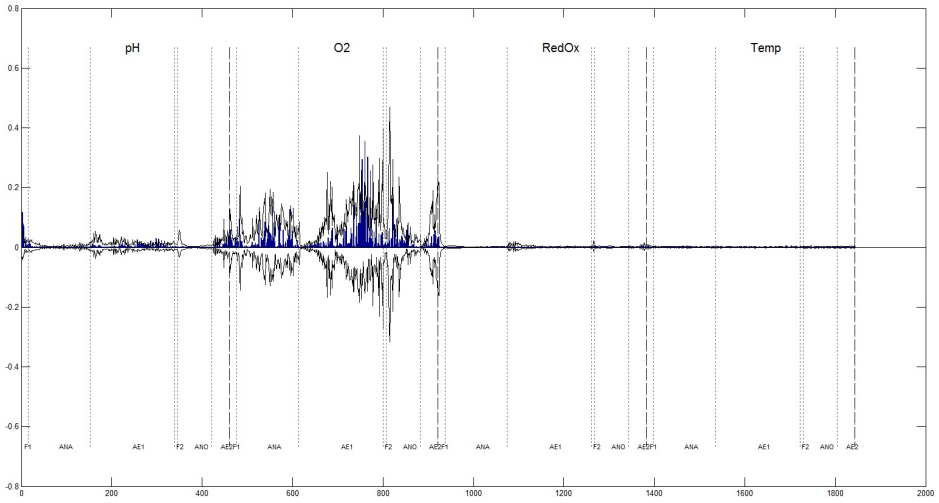


Figura 35: PlotContLimQ/T2

Paràmetres d'entrada:

i — El numero de batch o el grup de batch que volem graficar

Paràmetres de sortida:

Void

plotContLimQ(double[] : i) : Void

Mostrar es el batch o llista de batch que volem veure. Alguns exemples vàlids serien 1, [1 2 3 4], 1:5, : ...

El resultat serà un gràfic semblant al de la Figura35. Podem observar que es mostren les contribucions al límit de Q i es subdivideix en Fases (part superior), variables (part inferior).

Paràmetres d'entrada:

i	El numero de batch o el grup de batch que volem graficar

Paràmetres de sortida:

Void

plotprincipals(double a, double b) : Void

Mostra la component principal "a" contra la component principal "b" també mostra la zona de variància típica (en gris). El resultat serà semblant al de la Figura36.

Paràmetres d'entrada:

a	El numero de component principal que volem graficar en l'eix de les abscisses
b	El numero de component principal que volem graficar en l'eix de les ordenades

Paràmetres de sortida:

Void

plotT2Q() : Void

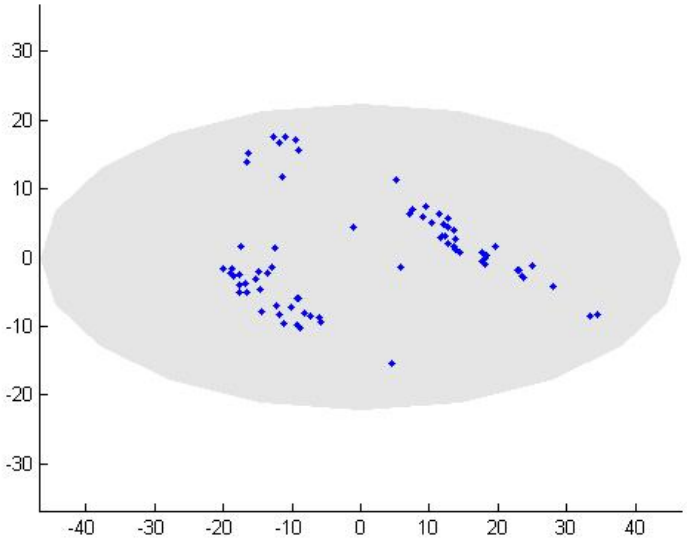


Figura 36: plotprincipals(a,b)

Mostra els límits de T2 i Q (línies negres) i tots els lots (color diferent segons la classe) en l'espai T2 contra Q on Q es a l'eix de les ordenades i T2 al de les abscisses. El resultat serà semblant al de la Figura37.

Paràmetres d'entrada:

Void

Paràmetres de sortida:

Void

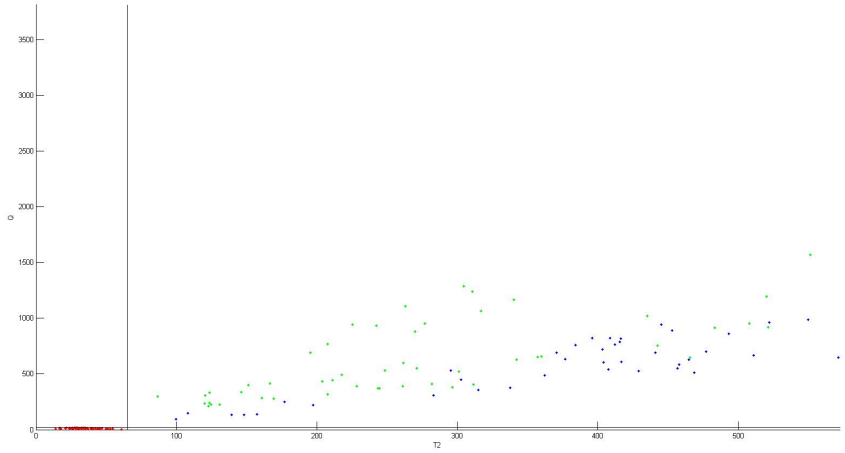


Figura 37: plotT2Q()

D.2 Funcions Privades

En aquest apartat explicarem que fa cada una de les funcions ocultes a l'usuari de la classe PCA.

CDKaiser() : Void

Aplica el criteri de Kaiser per fixar el numero de components principals que agafarem.

Grafscree() : Void

Mostrem el gràfic i demanem a l'usuari que ens entri el punt on es troba el colze.

VarExp() : Void

Calcula el nombre de components necessaris perquè la variança explicada sigui el entrat per l'usuari.

colzeanalitic() : Void

Necessitem mínim 3 variables per aplicar el metode.Busca el punt més proper a l'origen de coordenades (una aproximació de el colze).

estandaritza() : Void

Estandaritza la mostra. (funció per models).

estandaritz2() : Void

Estandaritza la mostra amb la mitja i std que tinguem entrada en aquest moment (funció per projectats).

calculaE() : Void

Calcula la matriu d'Errors.

calculaLimQ() : Void

Mètode per calcular el límit de Q.

calculaLimQv2() : Void

Mètode per calcular el límit de Q analíticament.

calculaLimT2() : Void

Mètode per calcular el límit de T^2.

calculaLimT2v2() : Void

Mètode per calcular el límit de T^2 analíticament.

calculacontT2() : Void

Mètode per calcular les contribucions de T^2.

calculacontQ() : Void

Calcula les contribucions a Q i el seu límit.

foraCQ() : bool[]

Retornara un vector amb els elements fora els límits de contribució per Q o -1 si es un model. El vector contindrà un 1 o un 0 per a cada una de les observacions, si a l'observació li correspon un 0 serà dins el model en canvi si li correspon un 1 serà una observació dolenta.

Paràmetres d'entrada:

Void

Paràmetres de sortida:

bool[] El vector contindrà un 1 o un 0 per a cada una de les observacions, si a l'observació li correspon un 0 serà dins el model en canvi si li correspon un 1 serà una observació dolenta.

plotLimCT2(double[] : i) : Void

Gràfic de barres amb contribucions al límit de T2.

Paràmetres d'entrada:

i El numero de batch o el grup de batch que volem graficar

Paràmetres de sortida:

Void

plotLimCQ(double[] : i) : Void

Gràfic de barres amb contribucions al límit de Q.

Paràmetres d'entrada:

i El numero de batch o el grup de batch que volem graficar

Paràmetres de sortida:

Void

Part VII

Apèndix III - CBR

E Notació

Diversos mètodes de l'objecte CBR utilitzen notacions en els paràmetres d'entrada per referir-se a característiques pròpies del funcionament de la classe. Ja sigui per repetició o extensió de les explicacions necessàries s'ha creat aquest apartat per no carregar d'informació l'explicació dels mètodes de la classe.

dc (Distance Criteria):

És el mètode que utilitzem per calcular les distancies entre els casos.

Actualment tenim definides 10 distancies diferents.

1 = Distància sobre les components principals

Les distancies en mode "1" les calculem com a distancia euclidiana on les dimensions són les diferents components principals de les dades, a la Figura38 hi tenim una representació gràfica de com es calcularien les distancies en mode "1".

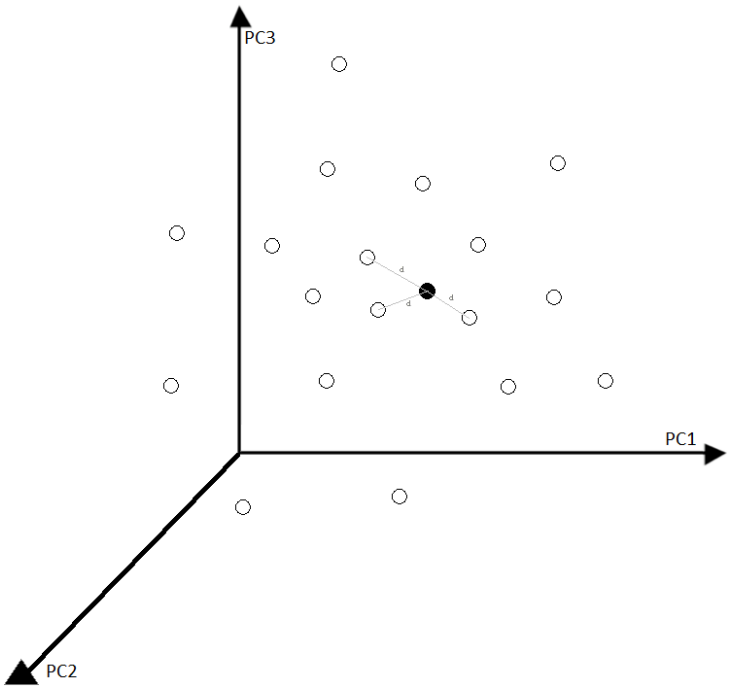


Figura 38: Distancia per Components Principals

$$d_t(c_a, c_b) = \sqrt{\sum_{i=1}^{r}(t_{c_a,i} - t_{c_b,i})^2} \tag{1}$$

On r serà el nombre de components principals del model, $t_{c_a,i}$ és la $i-esima$ classificació del cas c_a, per exemple una nova observació, i $t_{c_b,i}$ representaria el mateix per una observació en la base de casos.

2 = Distància sobre T^2

Les distancies en mode "2" les calculem com a distància euclidiana element a element de T^2.

A la Figura39 hi tenim una representació gràfica per entendre la idea de que representa la distancia.

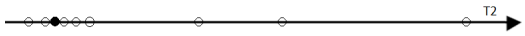


Figura 39: Similitud per T^2

3 = Distància sobre Q

Les distancies en mode "3" les calculem com a distància euclidiana element a element sobre l'espai Q.
A la Figura40 hi tenim una representació gràfica molt simple de la idea d'aquesta distancia.

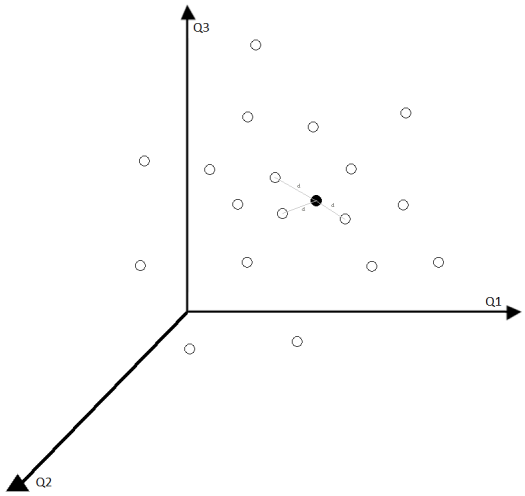


Figura 40: Similitud per Q

4 = Distància combinada sobre Q i T^2

Les distancies combinades ordenen els casos segons dues distancies distintes. Per fer ho primer fan la ordenació per un mètode de distancia i tallen segons el nombre de veïns escollit a "vk" per la primera distancia i seguidament d'aquests "vk(1)" veïns seleccionats segons el primer mètode de distancia, son ordenats segons el segon mètode. En fer el retreave es retornaran "vk(2)" casos. A la Figura41 hi trobarem la representació gràfica amb vk = [5 3].

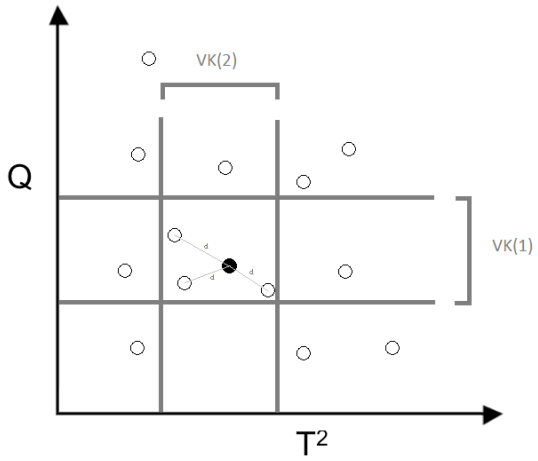


Figura 41: Distancia combinada per Q i T^2

5 = Distància combinada sobre Q i les principals components

Les distancies combinades ordenen els casos segons dues distancies distintes. Per fer ho primer fan la ordenació per un mètode de distancia i tallen segons el nombre de veïns escollit a "vk" per la primera distancia i seguidament d'aquests "vk(1)" veïns seleccionats segons el primer mètode de distancia, son ordenats segons el segon mètode. En fer el retreave es retornaran "vk(2)" casos. A la Figura42 hi trobarem la representació gràfica amb vk = [5 3].

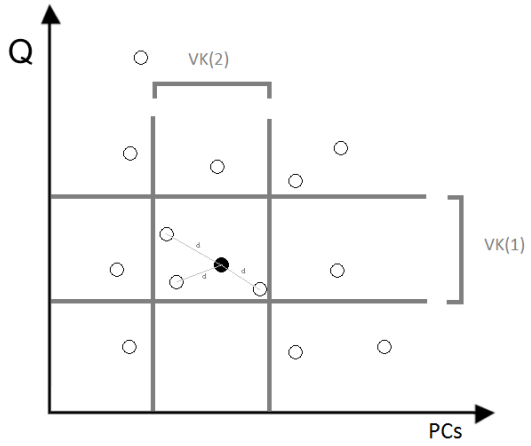


Figura 42: Distancia combinada per Q i les components principals

6 = Distància combinada sobre T^2 i les principals components

Les distancies combinades ordenen els casos segons dues distancies distintes. Per fer ho primer fan la ordenació per un mètode de distancia i tallen segons el nombre de veïns escollit a "vk" per la

primera distancia i seguidament d'aquests "vk(1)" veïns seleccionats segons el primer mètode de distancia, son ordenats segons el segon mètode. En fer el retreave es retornaran "vk(2)" casos. A la Figura43 hi trobarem la representació gràfica amb vk = [5 3].

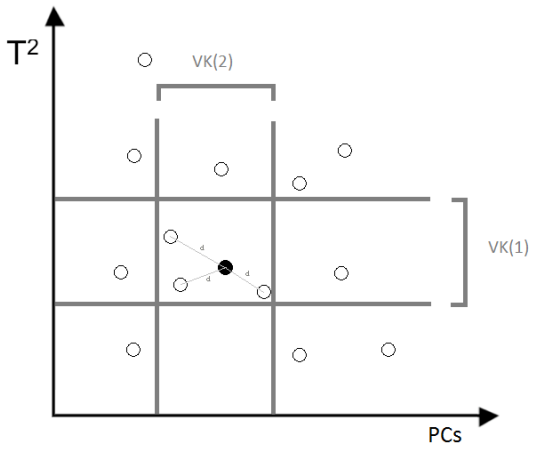


Figura 43: Distancia combinada per T^2 i les components principals

7 = Distància sobre Contribucions de Q en distancia euclidiana

8 = Distància sobre Contribucions de T^2 en distancia euclidiana

9 = Distància sobre Contribucions de Q en distancia mahalanobis

La utilitat d'utilitzar mahalanobis radica en la forma de determinar la similitud entre 2 variables aleatòries multidimensionals. Es diferencia de la distancia euclidiana en que te en compte la correlació entre les variables aleatòries. Ponderant segons la seva variància: les variables amb menys variància tindran més importància que les de major variància. D'aquesta manera es pretén igualar la importància de les variables amb la distancia.

10 = Distància euclidiana sobre Q i T^2

Tot i utilitzar Q i T^2 aquesta no es una distancia combinada, el que fara serà calcular les distancies en Q i en T^2 i calcular la distancia euclidiana ja que T^2 i Q son ortogonals.

vk:

És un vector de 2 elements que ens indica quants elements agafem per cada una de les distancies.Per exemple [7 3] indica que per la primera distancia agafarem els 7 elements més propers i posteriorment d'aquests 7 escollirem els 3 mes propers a l'element per fer-ne la classificació. En cas de tenir mètodes d'ordenació no compostos s'agafaran "min(vk)" elements. Tot i això si la base de casos no disposem de tants elements com s'ens demanen retornarem tants elements com ens sigui possible

theta:

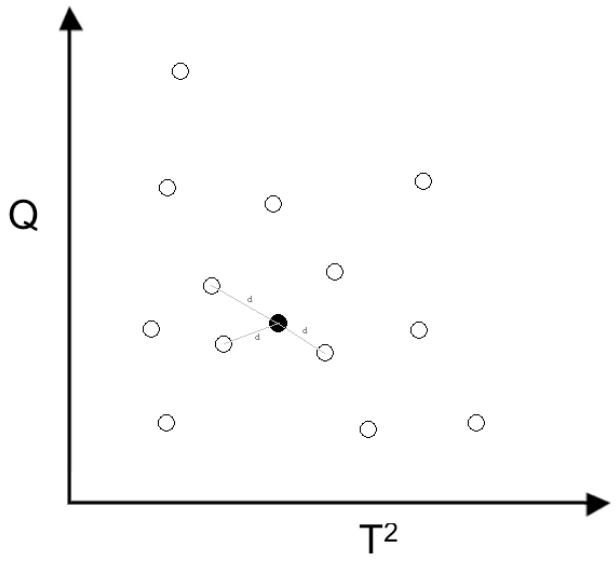


Figura 44: Distancia Euclidiana per Q i T^2

És la distancia que s'ha de complir entre dos casos per poder dir que son de la mateixa classe.

"Threshold that the ratio between the distance of cases of the reference class with respect the sum of all distances has to surpass in order that the new cases is from the reference class".

method:

Mètode utilitzat per determinar la classe dels nous casos (1 = Similitud(veí més proper), 2 = Voting, 3 = Voting ponderat per distancies).

F Atributs

Els atributs de la classe CBR són privats. Tenir els atributs privats ens permet controlar el tipus i la coherència de les dades que hi tenim en tot moment.

PCA : Model

Aquest camp serà un objecte de tipus PCA i fara la funció de model del nostre problema.

PCA : Case_Base

És un objecte de tipus PCA, el crearem projectant-lo sobre l'objecte Model, per tant les dades dels 2 objectes han de tenir les mateixes dimensions 1 i 2 (temps i variables). Aquest objecte serà la nostre base de casos sobre el problema.

bool : Case_Base_valid

És una indicador que ens serveix per saber si l'objecte Case_Base s'ha de projectar o no.

struct : veins

Aquí és on guardarem l'estructura "kNN" pels veïns. Aquesta estructura te 4 camps:

indexs_k1 = És l'ordre dels veïns de cada cas segons el criteri d'ordenació 1

distancia_k1 = És una matriu amb les distancies entre tots els casos segons el criteri d'ordenació 1

indexs_k2 = És l'ordre dels veïns de cada cas segons el criteri d'ordenació 2

distancia_k2 = És una matriu amb les distancies entre tots els casos segons el criteri d'ordenació 2

Si la distancia és simple trobarem que indexs_k1 = indexs_k2 i distancia_k1 = distancia_k2

PCA : test

Aquest camp és un objecte de tipus PCA on temporalment hi guardarem l'últim set de dades tractat o a tractar en breu (test set).

int [] : classepredit

Aquest camp és on emmagatzemarem temporalment la classe predita al fer el "Retrieve" per no haver d'obligar a l'usuari a passar-lo per paràmetre al "Reuse"...

bool : block

Aquest camp l'utilitzem per indicar que els paràmetres Vk,dc,thetha i method estan bloquejats perquè s'ha fet una optimització de la base de

casos.

G Funcions

G.1 Funcions Publiques

Tot seguit explicarem que fa cada una de les funcions a les que te accés l'usuari de la classe CBR.

G.1.1 Getters

Aquestes funcions són per definició les que ens permetran extreure informació de la nostra instància de l'objecte CBR.

getModel() : PCA

Aquest mètode ens retornara el model del nostre problema. Aquest camp pot no estar inicialitzat, si es el cas retornara null.

Paràmetres de sortida:

PCA	És el model del nostre problema.

getCase_Base() : PCA

Aquest mètode retorna la base de casos que s'esta fent servir en aquest moment a l'objecte actual. Retorna la base de casos com a objecte de tipus PCA, tal i com la guardem dins el nostre objecte CBR. Aquest

camp pot no estar inicialitzat, si es el cas retornara null.

Paràmetres de sortida:

PCA	És la base de casos del nostre problema.

G.1.2 Setters

Aquestes funcions són per definició les que ens permetran introduir informació a la nostra instància de l'objecte CBR.

setModel(PCA : model)

Aquest mètode ens servira per fixar el model al nostre objecte CBR, el paràmetre model ha de ser un objecte del tipus PCA.

Paràmetres d'entrada:

model	Serà el nou model del nostre CBR.

setCase_Base(PCA : CB)

Mètode per fixar una base de casos al nostre CBR, les dades d'aquest PCA són les que formen la nostra base de casos. El paràmetre CB ha de ser de tipus PCA.

Paràmetres d'entrada:

CB	És la nova base de casos del nostre objecte CBR.

G.1.3 Altres

Aquestes són les funcions especifiques per l'objecte CBR.

retrieve(double[][] dades,double[] vk,double dc) : struct [double[][] dades, Dades Test_set, double[] class]

Aquest mètode consisteix en recuperar un problema anterior de la Base de Casos que sigui semblant o igual al nou problema. El problema recuperat consta de dues parts, la descripció del problema més la seva la solució.

Els paràmetres Vk i dc són opcionals. Nomes cal introduir-los al sistema un cop, si no es passen per paràmetre assumirem que són els mateixos de les crides anteriors.

Paràmetres d'entrada:

dc	El mètode de distancia amb el que volem treballar.
vk	El numero de veïns que utilitzarem per fer el retrieve per cada un dels mètodes de distancia.
dades	Es l'únic paràmetre obligatori, correspon a les dades del nou problema.

Paràmetres de sortida:

struct	dades - Les dades dels casos mes semblants
	Test_set - dades del cas entrat projectades al model
	class - Les classes dels casos retornats

reuse(double theta,double method) : double[]

Basant-se amb la hipòtesis "Problemes similars tenen solucions similars" aquest mètode assigna una solució al nou problema "reutilitzant" la informació de la base de casos.

Els paràmetres theta i method són opcionals. Nomes cal introduir-los al sistema un cop, si no es passen per paràmetre assumirem que són els mateixos de les crides anteriors.

Paràmetres d'entrada:

method	Mètode utilitzat per escollir la classe del nou problema.
thetha	

Paràmetres de sortida:

double[]	La classe predita per cada un dels problemes que ens han demanat solucionar.

revise() : struct [double[][] matri,index[] fallen]

Aquesta funció s'encarregara de decidir si cal afegir el nou cas i la seva solució a la base de casos per futures utilitzacions.

retain(index[] fallen) : Void Afegeix el nou problema i la seva solució a la base de casos.

IB_UdG(double nelem,double[] vk,double dc,double theta,double method) : Void

Mètode per netejar la base de casos. Aquest mètode intentara definir les fronteres de les classes i despoblar els centres. Per fer-ho de cada element de la Base de casos actual afegim a la base de casos resultant el cas amb classificació distinta més proper. Aquest mètode tendeix a guardar molt de soroll.

Els paràmetres Vk , dc, theta i method són opcionals. Nomes cal introduir-los al sistema un cop, si no es passen per paràmetre assumirem que són els mateixos de les crides anteriors.

Paràmetres d'entrada:

dc	El mètode de distancia amb el que volem treballar.
vk	El numero de veïns que utilitzarem per fer el retrieve per cada un dels mètodes de distancia.
method	Mètode utilitzat per escollir la classe del nou problema.
thetha	
nelem	Numero de "enemics" que volem afegir per cada element a la nova base de casos. el valor òptim sol ser el mínim de components que s'utilitzen per calcular el retrieve. Un valor inferior disminuirà la base de casos resultant però provocara més errors de classificació, per altra banda un valor superior provoca que el tamany de la base de casos resultant augmenti i disminueixin els errors de classificació.

Paràmetres de sortida:

Void

IB3(double[] vk,double dc,double theta,double method) : Void

Mètode per netejar la base de casos proposat per DAVID W. AHA . Aquest mètode intentara definir les fronteres de les classes. Per fer-ho intenta separar les fronteres.L'algorisme implementat està basat en el codi de la Figura45.

Els paràmetres Vk , dc, theta i method són opcionals. Nomes cal introduir-los al sistema un cop, si no es passen per paràmetre assumirem que són els mateixos de les crides anteriors.

IB2(double[] vk,double dc,double theta,double method) : Void

Mètode per netejar la base de casos proposat per DAVID W. AHA . L'algorisme implementat està basat en el codi de la Figura46.

Els paràmetres Vk , dc, theta i method són opcionals. Nomes cal introduir-los al sistema un cop, si no es passen per paràmetre assumirem que són els mateixos de les crides anteriors.

DROP4(double[] vk,double dc,double theta,double method) : Void

Mètode per netejar la base de casos proposat per Wilson i Martinez. És un procés decremental, comença amb la base de casos completa i en treu els que no són necessaris. A la Figura47 tenim l'algorisme proposat per Wilson i Martinez, l'hem reproduït tan fidelment com ens ha estat possible.

Els paràmetres Vk , dc, theta i method són opcionals. Nomes cal introduir-los al sistema un cop, si no es passen per paràmetre assumirem que són els mateixos de les crides anteriors.

```
The IB3 algorithm (CD = Concept Description)
CD = []
For each x in TrainingSet do
    For each y in CD do
        Sim[y] = Similarity(x, y)
    end
    if Existsy in CD | acceptable(y) then
        ymax = some acceptable y in CD with maximal Sim[y]
    else
        i = randomly−selected value in CD
        ymax = some y in CD that is the i−th most similar instance to x
    end
    if class(x) = class(ymax) then
        classification ≪- correct
    else
        classification ≪- incorrect
        CD ≪- CD U [x]
    end
    For each y in CD do
        if Sim[y] > Sim[ymax] then
            Update y's classification record
            if y's record is significantly poor then
                CD ≪- C − [y]
            end
        end
    end
end
```


Figura 45: Codi IB3 [Aha, 1991, 1992]

```
The IB2 algorithm (CD = ConceptDescription)
CD = []
for each x in Training Set do
   for each y in CD do
      Sim[y] = Similarity(x, y)
   end
   ymax = Max(Sim[])
   if class(x) = class(ymax)
      classification ≪- correct
   else
      classification ≪- incorrect
       CD = CD U [x]
   end
end
```

Figura 46: Codi IB2 [Aha, 1991, 1992]

```
DROP4(Training Set = T): Instance set S
Let SI = T
//inicialize the lists of neighbours and associates
For each instance i in S
    Make sure we know the neighbors of i in S
    Add i to each of its neihbors' lists of associates
//Do careful noise-filtering pass  For each instance i in S
  if i is misclassified by its neighbors
     removeifnothelping(i,S)
//Do more aggresive reduction pass  Sort instances by distance to nearest enemy (furthest ones first)
For each instance i in S(starting with those furthest from their nearest enemy)
removeifnothelping(i,S)
Return S
```

Figura 47: Decremental Reduction Optimization Procedure 4 (DROP4) (Wilson and Martinez 2000)

Part VIII

Apèndix IV - Format CSV

G.2 Descripciò

Aquest format de fitxers ha estat pensat per introduir de manera ràpida tots els camps als objectes de tipus Dades. L'estructura del fitxer és molt simple s'utilitzen comes per separar els valors i com a delimitador de fi de línia s'utilitza /n/r.
Estructura d'un fitxer en format CSV:

Nom, 'El nom del proces'
Variables, 'Numero de variables'
'El nom de les variables separades per comes'
Fases, 'numero de fases'
'el nom de les fases separades per comes'
Duracio de Fases, 'numero de fases'
'la duracio de les diferents fases separades pe comes'
ref. Lots, 'numero de lots'
'el nom de cada un dels lots'
class, 'numero de lots'
'la classe de cada un dels lots'
data, 'instants de temps'**,** 'numero de variables'**,** 'numero de lots'
'les dades en format Variable - Wise'

Tots els camps que s'esmenten són obligatoris, en cas de no tenir-ne caldrà indicar com a numero 0 i deixar la línia on van els valors en blanc.

G.3 Exemple

```
Nom,Exemple de Fitxer CSV
Variables,4
pH,O2,RedOx,Temp
Fases,6
F1,ANA,AE1,F2,ANO,AE2
Duracio de Fases,6
15,138,187,6,75,40
ref. Lots,0

class,59
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
data,461,4,59
6.764,0.017974,80.031,19.406
6.7419,0.017463,86.497,19.305
6.7884,0.014336,64.933,19.205
.
.
.
```