Aprendizaxe automática
Exercicios para as clases interactivas

Tema 1: Clasificación e regresión

Recomendo executar os exercicios das clases interactivas nun sistema operativo Linux usando as linguaxes de programación Octave, Python 3, R e Java con Weka 3.8.5.

1. Programas en Octave

Imos comezar cos problemas de clasificación `hepatitis` (2 clases) e `wine` (3 clases), ambos proporcionados pola UCI Machine Learning Repository. A magnitude a predecir é a primeira columna do arquivo. Asemade, descarga o arquivo `airfoil.data`, que corresponde co problema de regresión da UCI nomeado **Airfoil self noise**. Canto a software, instala o `Octave`. Para algúns programas necesitarás o `Matlab`. Recomendo usar o editor `Kate` para escribir os programas, e executar o Octave nunha terminal de comandos, no canto do seu entorno gráfico.

1.1. Validación sobre o conxunto de entrenamento

1. Escribe un programa de Octave chamado `knn_id.m` que cargue dende un arquivo un problema de clasificación completo (usa primeiro o `hepatitis`, de 2 clases, e logo o `wine`. O programa debe calcular a saída `z` do clasificador para o propio conxunto de entrenamento. Os patróns deben pre-procesarse para ter media 0 e desviación 1. O programa debe entrenar o clasificador 1NN sobre un problema de clasificación completo. Usa a seguinte función `dist2.m` de Octave, que retorna a matriz coas distancias entre dúas matrices de datos `x` e `y`:

```
function d2=dist2(x,y)
  [Nx,nx]=size(x);[Ny,ny]=size(y);
  if nx~=ny
    error('matrices de anchos distintos')
  end
  d2=(ones(Ny,1)*sum((x.^2)',1))'+ones(Nx,1)*sum((y.^2)',1)- ...
    2.*(x*(y'));
end
```

2. Escribe unha función en Octave nomeada `knn` que reciba unha matriz `tx` cos patróns de entrenamento, un vector columna `y` coas saídas verdadeiras, unha matriz `sx` cos patróns de teste, e o número V de veciños máis cercanos a considerar. Para isto, debes facer unha votación entre os V veciños máis cercanos do patrón de teste $\mathbf{x}$ para obter a clase predita $y(\mathbf{x})$. A función debe retornar un vector columna `z` coas saídas preditas para os patróns de teste. Usa a función `dist2` do exercicio anterior.

3. Escribe unha función `matriz_confusion()` que retorne a matriz de confusión a partir das clases verdadeiras `y` e preditas `z`. O programa `knnc_id.m` debe chamar a esta función, mostrar a matriz de confusión por pantalla e calcular o acerto en %.

4. Escribe unha función `calcula_kappa()` que calcule, a partir das clases verdadeira `y` e predita `z`, o kappa en %. O programa `knnc_id.m` debe chamar a esta función e mostrar por pantalla o seu valor. Executa este programa cos problemas `hepatitis` e `wine`.

5. En caso de tratarse dun problema de dúas clases, o programa `knnc_id.m` debe calcular a sensitividade e especificidade (en %), xunto coa precisión e F-score con β=1.

6. Descarga o arquivo `tsa.data`, cos datos do problema de clasificación 2D *Two-spirals-appart*. Este problema de clasificación ten N=201 patróns bi-dimensionais (n=2) e C=2 clases $j = 1, 2$, con M=101 patróns/clase. As coordenadas $\{x_{ji}, y_{ji}\}$ dos patróns son $x_{ji} = r_i \cos t_{ij}$ e $y_{ji} = r_i \operatorname{sen} t_{ij}$, onde $i = 0, \ldots, M-1$. Ademáis, $r_i = ui$ e $t_{ij} = ui + j$, sendo $u_i = \dfrac{4\pi i}{M}$. Escribe o seguinte programa `plot_2d_clasif_map.m`, que aplica un clasificador (p. ex., `knn` ou `lda`) a este problema e crea a gráfica da figura 1, coa representación 2D dos patróns de entrenamento (panel esquerdo) e co mapa de clases aprendido polo clasificador (panel dereito). Este programa pode ser usado para representar o mapa de clasificación creado por calquera clasificador e problema de clasificación 2D.

```
clear all;clc
clasif='knn';
% lectura de datos-----------------------------------
dataset='tsa';
dir1='../data';nf=sprintf('%s/%s.data',dir1,dataset);
x=load(nf);y=x(:,end);x(:,end)=[];
for i=1:2
  t=x(:,i);x(:,i)=(t-min(t))/range(t);
end
figure(1);clf;subplot(1,2,1)
i=find(y==1);plot(x(i,1),x(i,2),'bs','markerfacecolor','b')
hold on
i=find(y==2);plot(x(i,1),x(i,2),'ro','markerfacecolor','r')
grid on
title(sprintf('Dataset %s',dataset))
% creacion de patrons de teste
t=x(:);vmin=min(t);vmax=max(t);
l=100;t=linspace(vmin,vmax,l);k=1;x2=zeros(l^2,2);
for i=1:l
  u=t(i);
  for j=1:l  % 1-t(j) para invertir o senso do eixo vertical
    x2(k,1)=u;x2(k,2)=1-t(j);k=k+1;
  end
end
% clasificacion de patrons de teste
switch(clasif)
case 'lda'
  z2=lda(x,y,x2);
case 'knn'
  z2=knn(x,y,x2,1);
otherwise
  error('clasificador %s non atopado',clasif)
end
% representacion das clases
subplot(1,2,2);map=zeros(l,l,3);k=1;
for i=1:l
  for j=1:l
    if z2(k)==1
      map(j,i,3)=0.8; % clase 1: azul
    else
```

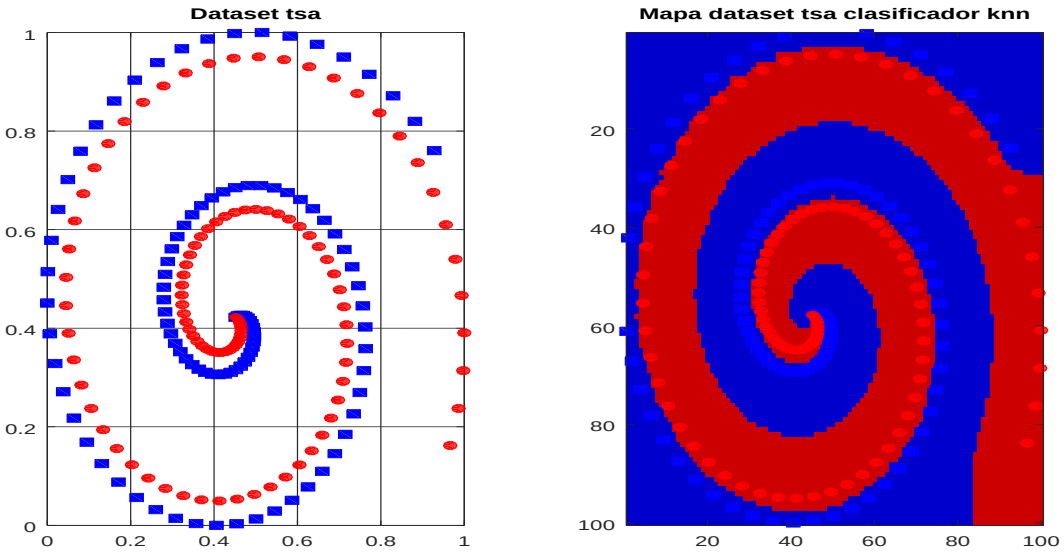

Figura 1: Problema de clasificación 2D *Two spirals appart* (esquerda) e mapa das clases creadas polo clasificador 1NN neste problema (dereita) creado polo programa `plot_2d_clasif_map.m`.

```
        map(j,i,1)=0.8; % clase 2: vermello
      end
      k=k+1;
    end
end
imagesc(map);x=1*x;hold on
i=find(y==1);plot(x(i,1),1-x(i,2),'bs','markerfacecolor','b')
hold on
i=find(y==2);plot(x(i,1),1-x(i,2),'ro','markerfacecolor','r')
title(sprintf('Mapa dataset %s clasificador %s',dataset,clasif))
nf=sprintf('mapa_%s_%s.eps',clasif,dataset);print('-depsc',nf)
```

7. Escribe outro programa de Octave chamado `knnr_id.m`, que aplique o método 1NN ao problema de regresión `airfoil`. Neste caso, o programa debe calcular a medida de calidade R e as medidas de erro RMSE e MAE.

8. Engade o código para que o programa cree o diagrama de dispersión (*scatterplot*) cos valores verdadeiro e predito de `y`, como se mostra na figura 2. Garda este gráfico nun arquivo en formato EPS (fai isto tamén en tódalas gráficas que se fagan máis adiante).

9. Crea tamén o diagrama cos valores verdadeiros e preditos da figura 3:

1.2. Validación cruzada con sintonización dos hiper-parámetros

1. Crea unha función de Octave nomeada `crea_folds()` que reciba como argumentos a matriz `x` cos patróns, o vector columna `y` coas clases verdadeiras, e o número K de probas da validación cruzada. A función debe retornar as matrices `tx`, `vx` e `sx` cos patróns de entrenamento, validación e teste, respectivamente, xunto cos vectores columna `ty`, `vy` e `sy` coas clases verdadeiras para os patróns de entrenamento, validación e teste, respectivamente, e para cada proba $k = 1..K$ de validación cruzada. Para isto, a función debe seleccionar, para cada clase $i = 1, \ldots, C$, os patróns pertencentes á devandita clase, xerar un ordeamento aleatorio reproducíbel destos patróns, e asignar estos patróns, para cada proba de validación cruzada, aos conxuntos de entrenamento, validación ou teste que corresponda.

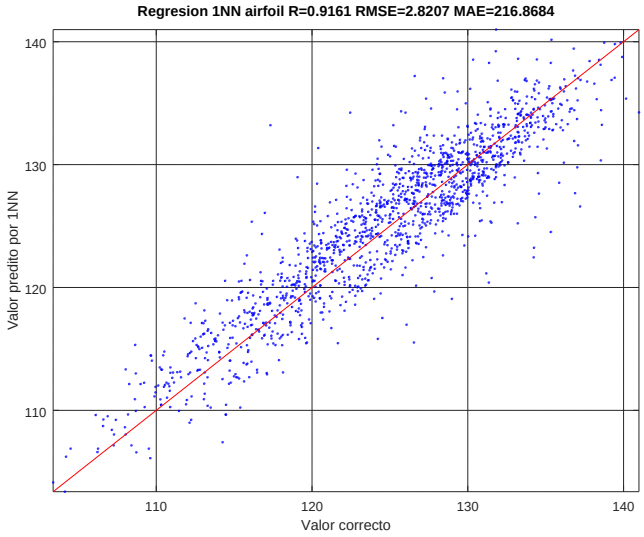

Figura 2: Scatterplot cos valores verdadeiro e predito por NN para o dataset `airfoil` creado polo programa `knnr_id.m`.

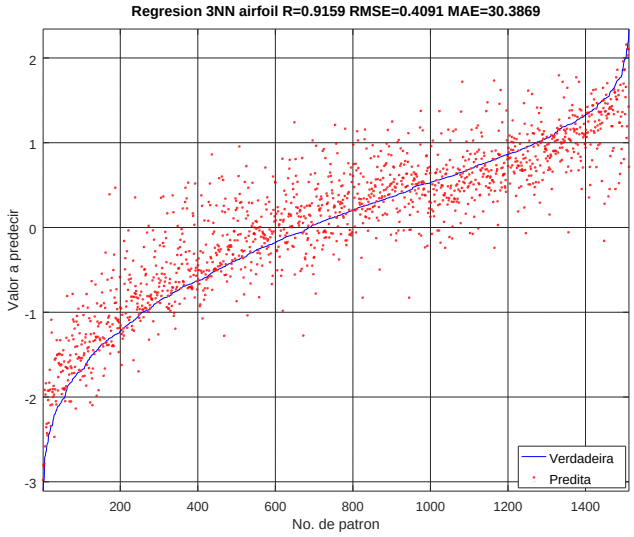

Figura 3: Diagrama cos valores verdadeiro e predito para o dataset `airfoil`.

2. Crea un programa `knnc.m` que cargue un problema de clasificación e chame á anterior función `crea_folds()` para xerar os conxuntos de entrenamento, validación e teste. O programa debe pre-procesar os tres conxuntos restando a media e dividindo pola desviación típica dos datos de entrenamento.

3. O programa `knnc.m` debe sintonizar o hiper-parámetro V (nº de veciños máis cercanos) do clasificador KNN. Para isto, debes usar valores de V no conxunto $\{1, 3, 5, 7, 9, 11\}$ (sempre valores impares para evitar empates). Para cada valor de V, debes entrenar o clasificador no conxunto de entrenamento e validalo no conxunto de validación. Isto repítese para os K folds, e calcúlase o kappa medio obtido por NN con ese valor de V. Repite o proceso para tódolos valores de V e selecciona o valor que maximiza o kappa. Crea unha gráfica como a figura 4 que mostre o comportamento de kappa frente a V.

4. Introduce neste mesmo programa `knnc.m` a etapa de teste, na cal repites o entrenamento e validación sobre as K particións de validación cruzada. O entrenamento na proba k-ésima debe

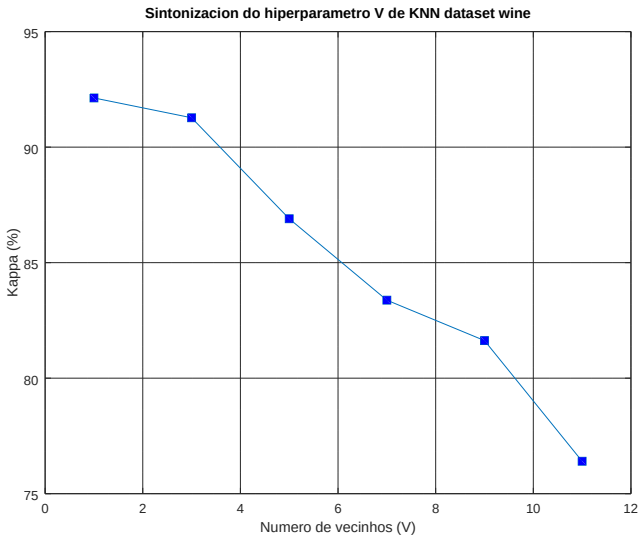

Figura 4: Diagrama co kappa medio para os distintos valores de V en KNN para o dataset `wine`.

usar os conxuntos de entrenamento e validación desa proba. O teste debe usar a partición de teste. O valor final de kappa, acerto e matriz de confusión deben ser o promedio sobre as K probas. Na clasificación binaria ($C = 2$ clases), calcula tamén a precisión e F-score con β=1 a partir da matriz de confusión promedio. Proba o programa cos problemas `hepatitis` e `wine`.

5. Crea unha función `crea_folds_reg` que realice un papel análogo á anterior función `crea_folds` pero para problemas de regresión. Para isto, ordea as saídas verdadeiras e logo realiza unha selección aleatoria reproducíbel. Finalmente, reparte éstas entre os tres conxuntos (entrenamento, validación e teste) para cada proba.

6. Crea un programa `nnr.m` que faga o mesmo que `knnc.m` pero para un problema de regresión usando a función anterior `crea_folds_reg`, aplicándoo ao problema `airfoil`. Debes calcular as mesmas medidas de calidade e elaborar os mesmos gráficos que no apartado 1.

2. Execución en Python

Para executar clasificadores e regresores en Python, debes instalar e importar o módulo Scikit-learn. Recomendo executar os programas dende o intérprete `ipython3`, que podes executar co comando:

```
ipython3 --pylab
```

para que incorpore xa algúns módulos como o `numpy` para a execución de comandos.

2.1. Validación sobre o conxunto de entrenamento

1. Escribe un programa `nn1.py` que use o obxecto `KNeighborsClassifier` do módulo `sklearn.neighbors` para entrenar e testear un clasificador KNN con V=5 veciños sobre o conxunto de entrenamento completo. Para isto, debes usar os seguintes métodos:

```
modelo=KNeighborsClassifier(n_neighbors=V).fit(x,y) # entrenamento
z=modelo.predict(x)  # teste
```

2. Mostra por pantalla o kappa, acerto e matriz de confusión usando as funcións `cohen_kappa_score(y,z)`, `accuracy_score` e `confusion_matrix`, todas cos mesmos argumentos e pertencentes ao módulo `sklearn.metrics`.

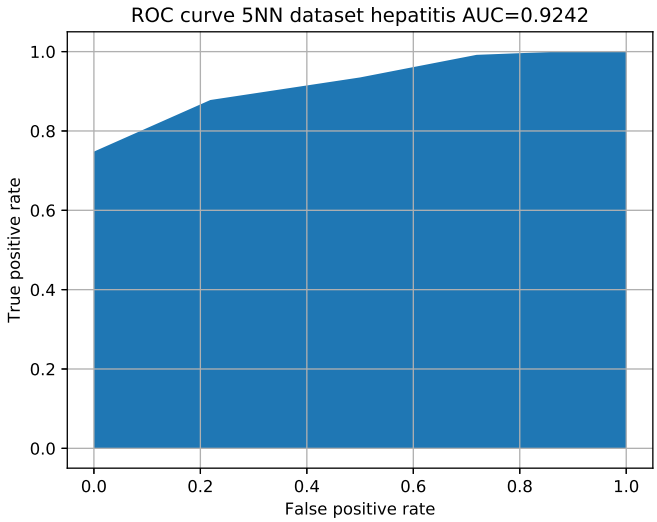

Figura 5: Curva ROC e AUC do clasificador 5NN no dataset `hepatitis` creada polo programa `nn1.py`.

3. En problemas de clasificación binaria (problema `hepatitis`), calcula a precisión (función `precision_score`), recall (`recall_score` e F1 (`f1_score`), cos mesmos argumentos e módulo que no exercicio anterior.

4. En clasificación binaria, crea un gráfico coa curva ROC como o da figura 5 e calcula a AUC usando a función `roc_auc_score`. Usa o método `predict_proba` do obxecto `KNeighborsClassifier` para calcular as probabilidades de clase para os patróns de entrenamento.

```
aux=modelo.predict_proba(x);y2=y-1
p=aux[:,1] # probabilidade da clase 1
fpr,tpr,umbrais=roc_curve(y2,p)
from matplotlib.pyplot import *
clf(); #plot(fpr,tpr,'bs--')
n=len(fpr);X=range(n);Z=zeros(n);fill_between(fpr,zeros(n),tpr)
ylabel('True positive rate')
xlabel('False positive rate')
title('ROC curve %iNN dataset %s AUC=%.4f' \
  %(V,dataset,roc_auc_score(y2,p)))
grid(True);show(False)
savefig('roc_curve_%inn_%s.eps'%(V,dataset))
```

2.2. Validación cruzada con sintonización dos hiper-parámetros

1. Escribe un programa `nn2.py` que divida o problema nun conxunto de entrenamento e outro de teste usando a función `train_test_split` do módulo `sklearn.model_selection`. Logo, usa o obxecto `GridSearchCV`, do módulo `sklearn.model_selection`, para sintonizar o hiper-parámetro V con valores $\{1, 3, 5, 7, 9, 11\}$ optimizando a F1 e mostrando o seu valor para cada V.

2. No mesmo programa `nn2.py` usa a función `cross_val_predict` do módulo `sklearn.model_selection` para calcular a saída do clasificador na validación cruzada 4-fold, usando o mellor valor de V seleccionado no exercicio anterior. Calcula as medidas de calidade (kappa, acerto, matriz de

confusión e medidas para clasificación binaria, no seu caso). Executa o programa cos dous problemas de clasificación.

3. O programa `nn2.py` do exercicio anterior proporciona unha estimación irreal e optimista da operación do clasificador, xa que éste avalíase realizando unha validación cruzada onde os conxuntos de teste inclúen patróns xa usados para sintonizar o hiper-parámetro V. Para facer unha estimación realista, hai que usar conxuntos de entrenamento, validación e teste como nos exercicios co Octave. Crea unha función de Python nomeada `crea_folds()` nun arquivo `nn.py` que retorne as matrices `tx`, `vx` e `sx` cos patróns de entrenamento, validación e teste, respectivamente, para as K probas de validación cruzada, xunto cos vectores columna `ty`, `vy` e `sy` coas clases verdadeiras para os patróns de entrenamento, validación e teste, respectivamente.

4. Neste programa `nn.py`: chama á función `crea_folds()`; pre-procesa os tres conxuntos de datos, usando as medias e desviacións do entrenamento; sintoniza o hiper-parámetro V, tal como fixeches en Octave; selecciona o V que maximiza o kappa promedio sobre os K conxuntos de validación; e calcula as medidas de calidade promedio sobre os K conxuntos de teste, logo de entrenar cos patróns de entrenamento e validación. Executa o programa cos problemas `hepatitis` e `wine`.

5. Escribe un programa nomeado `nnr.py` que realice o mesmo proceso pero para problemas de regresión, usando o obxecto `KNeighborsRegressor` de do módulo `sklearn.neighbors`, e calcula as medidas de calidade e erro e as gráficas de regresión. Execútao co problema `airfoil`.

3. Execución en R

1. Escribe un programa en R (enlace) chamado `nn.r` que use a función `knn`, `knn1` (para 1NN) e `knn.cv` (para validación cruzada Leave-One-Out) do paquete `class` para clasificar os problemas `hepatitis` e `wine`. Podes comprobar se este paquete está instalado co comando `library(class)`. Se non está instalado, podes facelo co comando `install.packages('class')`. Usa a función `read.table` para ler os datos. Usa a función `table` para crear a matriz de confusión. Podes executalo dende R con `source('nn.r')`.

2. Escribe unha función de R nomeada `calcula_kappa` no mesmo programa anterior que calcule kappa a partir da matriz de confusión.

3. Executa o programa `nn.r` anterior co problema de regresión `airfoil`. Neste caso, a primeira columna da matriz de datos ten valores continuos e a saída `z` tamén, aínda que é de tipo `factor`. Por iso hai que convertila a número coa función `as.numeric(y)` antes de calcular R, $RMSE$ e MAE.

4. (Proposto) Crea un programa nomeado `caret.r` que use a función `train` do módulo `caret` de R, que debes instalar. Esta función é unha interface que permite executar de modo transparente moitos clasificadores e regresores de moitos paquetes de R. Además, indica que hiper-parámetros sintonizábeis ten cada algoritmo, e propón valores razoábeis para cada un deles.

4. Execución en Weka

Para executar un clasificador en Weka, descarga a última versión (3.8.5) de Weka.

1. Descarga o problema de clasificación `diabetes` da UCI, que está en formato ARFF de Weka. Executa o programa `NearestNeighbor.java`, que entrena e testea un clasifidor básico, neste caso IBk, que é o clasificador NN, sobre o problema `diabetes`.

```
// export WEKAINSTALL=~/weka-3-8-5
// export CLASSPATH=$WEKAINSTALL/weka.jar:.
// javac NearestNeighbor.java
// java NearestNeighbor
import weka.classifiers.Evaluation;
import weka.classifiers.lazy.IBk;
import weka.core.Instances;
import java.util.Random;
import java.io.BufferedReader;
import java.io.FileReader;

public class NearestNeighbor {
  static int  C;

  public static void printMatConf(double [][] mc) {
    int  i,j;
    System.out.printf("Matriz de confusion:\n");
    for(i=0;i<C;i++) {
      System.out.printf(String.format("\t%6d", i));
    }
    System.out.printf("\n");
    for(i=0;i<C;i++) {
      System.out.printf(String.format("%d", i));
      for(j = 0; j < C; j++) {
        System.out.printf(String.format("\t%6.0f", mc[i][j]));
      }
      System.out.printf("\n");
    }
  }
  public static void main(String [] args) throws Exception {
    Instances x=new Instances(new
      BufferedReader(new FileReader("../data/diabetes.arff")));
    double [][]  mc;
    double  kappa;
    System.out.printf("executando IBk en dataset diabetes...\n");
    x.setClassIndex(x.numAttributes()-1);
    C=x.numClasses();
    Evaluation eval = new Evaluation(x);
    IBk nn=new IBk();
    eval.crossValidateModel(nn,x,10,new Random(1));
    System.out.println(eval.toSummaryString("Results:", false));
    mc=eval.confusionMatrix();
    kappa=100*(float)eval.kappa();
    System.out.printf("kappa=%.1f%%\n",kappa);
    printMatConf(mc);
  }
}
```

Para empregalo, tes que executar:

```
export WEKAINSTALL=~/weka-3-8-5
export CLASSPATH=$WEKAINSTALL/weka.jar:.
javac NearestNeighbor.java
java NearestNeighbor
```

2. (Proposto) Modifica o anterior programa `NearestNeighbor.java` para que no canto de executar validación cruzada, o programa xenere as particións de entrenamento, validación e teste

como en Octave e Python, e execute validación cruzada K-fold, con K=4, sintonizando o hiper-parámetro V do clasificador IBk con valores V=1,3,5,7,9,11 e 13. Modifica tamén este programa para realizar regresión de veciños máis cercanos. Introduce tamén o preprocesamento media cero e desviación 1.

Tema 2: Análise discriminante linear e regresión linear

1. Programas en Octave

1. Escribe a seguinte funcion `lda` de Octave, que entrena un clasificador LDA sobre un conxunto de datos e retorna a clase predita para os patróns de teste. Tamén pode proporcionar as probabilidades das distintas clases para cada patrón.

```
% lda: implements the lda classifier
% output: z the predicted class
% inputs: x matrix with the training patterns (each pattern one row)
%         y vector with the desired output in training set
%         sx matrix with the test patterns
function z=lda(x,y,sx)
  [N, I]=size(x);
  cl=unique(y);C=numel(cl);
  nc=zeros(C,1);  % number of patterns per class
  mc=zeros(C,I);  % mean of each class
  S=zeros(I); % total covariance
  w=zeros(C,I+1); % coeficients of LDA
  for i=1:C
    j=(y==cl(i));nc(i)=sum(j);
    u=x(j,:);mc(i,:)=mean(u);
    S=S+(nc(i)-1)*cov(u)/(N-C);
  end
  pr=nc/N;  % pr=probabilities
  for i=1:C
    u=mc(i,:);t=u/S;
    w(i,1)=log(pr(i))-t*u'/2;  % offset
    w(i,2:end)=t;  %  linear term
  end
  T=size(sx,1);  % no. test patterns
  L=[ones(T,1) sx]*w';  % linear scores
  % implement softmax function
  P=exp(L)./repmat(sum(exp(L),2),[1 C]);  % class probabilities
  [~,z]=max(P,[],2);  % predicted class by LDA
end
```

---

2. Escribe un programa nomeado `ldac_id.m` que cargue os datos do problema de clasificación, chame á anterior función `lda` para que entrene e valide sobre o conxunto de datos completo. Execútao sobre os problemas `hepatitis` e `wine`.

3. Crea un programa chamado `ldac.m` que realice unha validación cruzada K-fold con K=4, sen sintonización de parámetros, baseándote para isto no programa `nn.m` e na función `crea_folds` do tema anterior.

4. Escribe un programa nomeado `linreg.m` que lea os datos dun problema de regresión, preprocese os datos para ter media cero e desviación 1 nas entradas e na saída, e realice unha regresión linear de mínimos cadrados empregando a fórmula matricial vista na clase expositiva para o cálculo do vector de pesos $\mathbf{w}$. Calcula as medidas de calidade e as gráficas propias da regresión para o programa co problema `airfoil`.

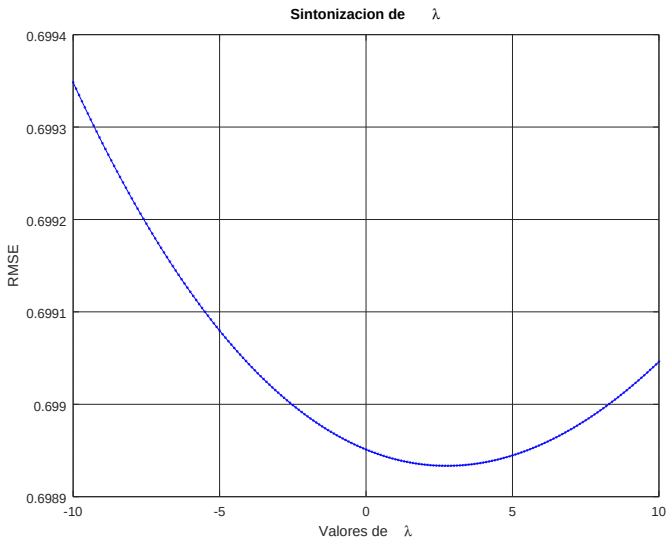

Figura 6: Gráfico coa sintonización do parámetro de regularización λ para a regresión regularizada no dataset `airfoil`, creado polo programa `ridge.m`.

5. Escribe un programa chamado `ridge.m` que realice regresión regularizada (*ridge regression*) sintonizando o hiper-parámetro λ (regularización) con valores entre -10 e 10 con paso 0.1. Xera a figura 6 para ilustrar este proceso de sintonización. Xera as saídas propias do problema de regresión `airfoil`.

2. Execución en Python

1. Escribe un programa nomeado `lda.py` que use o clasificador `LinearDiscriminantAnalysis` do módulo `sklearn.discriminant_analysis` para entrenar e validar sobre o conxunto de datos completo (con pre-procesamento incluído). Logo, o programa debe realizar unha validación cruzada 4-fold usando a función `cross_val_predict` do módulo `sklearn.model_selection`. O programa non debe incluír a sintonización de hiper-parámetros xa que o LDA non ten ningún hyper-parámetro sintonizábel.

2. Escribe un programa `linreg.py` que use o obxecto `LinearRegression` do módulo `sklearn.linear_model` para realizar regresión linear sobre o problema `airfoil`. O pre-procesamento debe incluír a saída.

3. Implementa a regresión regularizada en Python (programa `ridge.py`) usando o obxecto `Ridge` do mesmo módulo, usando a función `crea_folds` escrita no tema 1 para a validación cruzada K-fold e sintonizando o parámetro de regularización λ, chamado α na sintaxe deste obxecto, con valores entre -10 e 10 e paso 0.1.

3. Exercicios propostos

1. Implementa o LDA, regresión linear e regresión regularizada (ridge) en R, usando as funcións `lda`, `lm` e `lm.ridge` do paquete `MASS`.

2. Implementa o LDA e regresións linear a regularizada en Weka, usando o clasificador `weka.classifiers.` e o regresor `weka.Classifiers.Functions.LinearRegression`, respectivamente (o hiper-parámetro de regularización λ é a opción `-R` de `LinearRegression`).

Tema 3: Árbores de clasificación e regresión

1. Programas en Octave e Matlab

1. Crea un programa `ctree1.m` en Octave que avalíe unha árbore de clasificación usando validación cruzada, pero sen sintonización de parámetros. Para isto, debe chamar á función `ctree2` no arquivo `ctree2.m`, que implemente esta árbore usando o pseudo-código que podes atopar na seguinte URL:

   https://www.cs.cmu.edu/~bhiksha/courses/10-601/decisiontrees

2. Crea un programa `ctree.m` en Matlab que entrene e valide un árbore de clasificación, usando a función `fitctree` pertencence á Statistics and Machine Learning Toolbox, sobre o conxunto de datos completo, con preprocesamento. Calcula as medidas de calidade de clasificación. Calcula tamén o erro de validación cruzada coa función `crossval`. A sintaxe é a seguinte:

```
modelo=fitctree(x,y);  % entrena
view(modelo,'mode','graph');  % visualiza arbore
z=predict(modelo,x);  % teste
fprintf('erro=%.1f%%\n',100*modelo.cvloss);  % erro cv
modelo=crossval(modelo,'KFold',4); % validacion cruzada
fprintf('erro=%.1f%%\n',100*modelo.kfoldloss);  % erro kfold
```

3. Escribe un programa `rtree.m` análogo para regresión usando a función `fitrtree` no canto de `fitctree`.

2. Programas en Python

1. Escribe un programa `ctree.py` en Python usando o obxecto `DecisionTreeClassifier` do módulo `sklearn.tree` coa seguinte sintaxe:

```
modelo=DecisionTreeClassifier().fit(x,y)  # entrena
z=modelo.predict(x)   # test
# representacion grafica da arbore------
from graphviz import *
dot_data=export_graphviz(modelo,out_file='tmp.dot')
from os import system,remove
nf='arbore_clasif_%s.eps'%dataset
system('dot -Teps tmp.dot -o %s'%nf)
remove('tmp.dot')
```

   Usa o módulo `graphviz` para representar a árbore de clasificación creada, como se mostra na figura 7. Para isto, debes tamén instalar o paquete `graphviz` de Linux, que proporciona o comando `dot` para exportar a figura (neste caso, en formato `eps`). De todos modos, a utilidade real deste diagrama da árbore é reducida porque para problemas reais da árbores demasiado grandes.

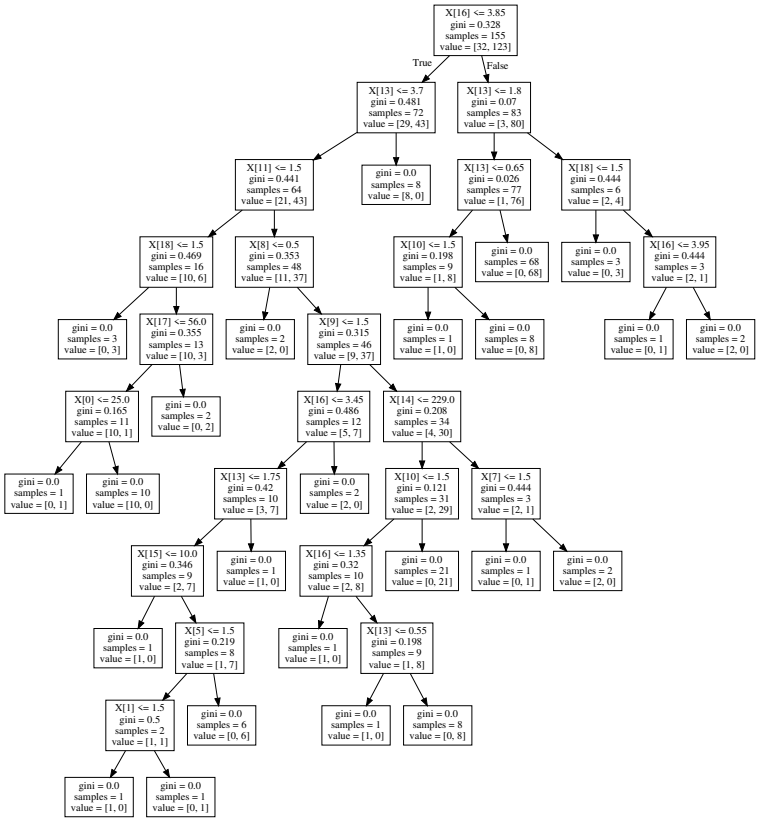

Figura 7: Árbore de clasificación creada por `ctree.py` para o dataset `hepatitis`.

2. Modifica o programa anterior para regresión, creando o programa `rtree.py` que use o obxecto `DecisionTreeRegressor` do mesmo módulo `sklearn.tree`.

3. Exercicios propostos

1. Implementa a árbore de clasificación e regresión en R, usando a funcións `rpart` (*recursive partitioning*) no paquete do mesmo nome.

2. Implementa a árbore de clasificación e regresión en Weka usando o obxecto `weka.classifiers.trees.J48`.

Tema 4: Redes neuronais

1. Programas en Octave

1. Escribe a seguinte función `mlpc` de Octave, que entrena un clasificador MLP sobre un conxunto de datos e retorna a clase predita para os patróns de teste. Usa as funcións `prestd` e `trastd` de Octave para preprocesar os patróns de entrenamento e validación. Usa funcións de activación tanxente hiperbólica nas neuronas das capas ocultas, e linear na capa de saída. Usa a función `newff` para crear un obxecto rede neuronal MLP, e as funcións `train` e `sim` para entrenar e testear a rede, respectivamente.

```
% mlp: implements the mlp classifier
% output: z the predicted output
% inputs: x matrix with training patterns (each pattern one row)
%   c vector with the desired output in training set
%   sx matrix with the test patterns
%   neurons a vector with the number of neurons of each layer
%   (the last layer is the output layer)
function z=mlpc(x,y,sx,neurons)
  nl=numel(neurons);tf=cell(1,nl); % tf=transfer function
  for i=1:nl-1; tf{i}='tansig'; end
  tf{nl}='purelin';
  mlp=newff(min_max(x'),neurons,tf,"trainlm",...
    "learngdm","mse");mlp.trainParam.show=inf;
  N=size(x,1);y2=zeros(neurons(end),N);
  for i=1:N
    j=y(i);y2(j,i)=1;
  end
  mlp=train(mlp,x',y2);
  y2=sim(mlp,sx');[~,z]=max(y2);
end
```

---

2. Escribe un programa `clasifMLP_id.m` que use a función do exercicio anterior para clasificar un problema entrenando e validando sobre o problema completo. Usa H=3 capas ocultas (sen contar a capa de saída). O número de neuronas ocultas de cada capa debe tomar valores entre 10 e 30 con paso 10. Executa o programa sobre os dous problemas de clasificación e calcula as medidas de calidade.

3. Escribe un programa `clasifMLP.m` que use a función `mlp` para entrenar unha rede MLP para clasificación sintonizando os hiper-parámetros H (número de capas ocultas) e H_i, con $i = 1, \ldots, H$ (número de neuronas na capa oculta i-ésima). Usa para isto a función `crea_folds` vista no tema 1, calcula as medidas de calidade e crea a gráfica de sintonización. Executa este programa para os problemas `hepatitis` e `wine`.

4. Modifica a función `mlpc` de máis enriba para crear unha función nova `mlpr` que entrene e testee o MLP para problemas de regresión.

5. Escribe un programa `regMLP.m` que use a función `mlp` para entrenar unha rede MLP para regresión con sintonización de hiper-parámetros, como no exercicio anterior. Usa para isto a función `crea_folds_reg` vista no tema 1. Calcula as medidas de calidade para regresión e crea as gráfica de sintonización e de regresión axeitadas. Executa este programa para o problema `airfoil`.

2. Programas en Python

1. Escribe un programa `mlpc.py` que use o obxecto `MLPClassifier` do módulo `sklearn.neural_network` para executar a rede MLP en problemas de clasificación con sintonización de hiper-parámetros.

2. Modifica o programa anterior, usando como referencia o programa `nnr.py`, para crear o programa `mlpr.py`, que usando o obxecto `MLPRegressor` para aprender un problema de regresión.

3. Exercicios propostos

1. Usa a función `nnet` do paquete `nnet` de R para executar unha rede MLP para clasificación e regresión.

2. Usa o obxecto `weka.classifiers.functions.MultilayerPerceptron` para executar a rede MLP para clasificación e regresión.

Tema 5: Redes profundas

1. Programas en Python

1. Instala o paquete de Linux `python3-keras`, asociado ao módulo de Python `keras`, que proporciona a rede Deep Learning. Este módulo pode executarse sobre `tensorflow` ou sobre `theano` (éste último está no paquete de Linux `python3-theano`).

2. Descarga os arquivos `mnist_train.dat` (60,000 patróns) e `mnist_test.dat` (10,000 patróns), do problema MNIST. Son imaxes de números escritos a man. Cada patrón é unha imaxe dun número con píxeles en escala de grises (784 valores entre 0 e 255). A clase é a primeira columna.

3. Crea un programa nomeado `dlc.py` que execute unha rede Deep Learning para clasificación. O número de neuronas da capa de saída debe ser igual ao número C de clases. A saída desexada para a rede debe ser unha matriz **ty** de orde $N \times C$ onde $ty_{ij} = 1$ so se o patrón i pertence á clase j. Calcula as medidas de calidade para clasificación e mide os tempos de entrenamento e teste usando a función `time` do módulo `time`. Esta rede é lenta entrenando (7.9 min. usando unha arquitectura con 2 capas ocultas con 50 e 10 neuronas sobre este problema).

```
Hi=[50,10];H=len(Hi)  # Hi=#capas ocultas,H=#neuronas/capa
modelo=Sequential()
modelo.add(Dense(Hi[0],kernel_initializer='uniform',\
  activation='relu',input_dim=n))  # capa de entrada
for h in range(1,H):  # H=no. capas ocultas
  modelo.add(Dense(Hi[h],kernel_initializer='uniform',\
    activation='relu'))
modelo.add(Dense(C,kernel_initializer='normal',\
  activation='softmax')) # capa saida
modelo.compile(optimizer='adam',loss='categorical_crossentropy',\
  metrics=['accuracy'])
modelo.fit(tx,ty,epochs=100,verbose=0)  # entrenamento
z=modelo.predict_classes(sx,verbose=0)  # validacion
```

4. Crea outro programa chamado `dlr.py` que execute rede Deep Learning the Keras no problema de regresión `airfoil`. Para isto, executa a validación cruzada 4-fold como no programa `nnr.py` usando a función `crea_folds`, aínda que sen sintonización de parámetros. Crea unha función `dlr()` que reciba `tx`, `ty` e `sx`, entrene a rede cunha arquitectura 50-10. Tes que cambiar as seguintes cousas na rede para que realice regresión:

```
modelo=Sequential()
modelo.add(Dense(Hi[0],kernel_initializer='normal',\
  activation='relu',input_dim=n))
for h in range(1,H):
  modelo.add(Dense(Hi[h],kernel_initializer='normal',\
    activation='relu'))
modelo.add(Dense(1,kernel_initializer='normal'))
modelo.compile(optimizer='rmsprop',loss='mse')
modelo.fit(tx,ty,epochs=100,verbose=0)
z=modelo.predict(sx,verbose=0)
```

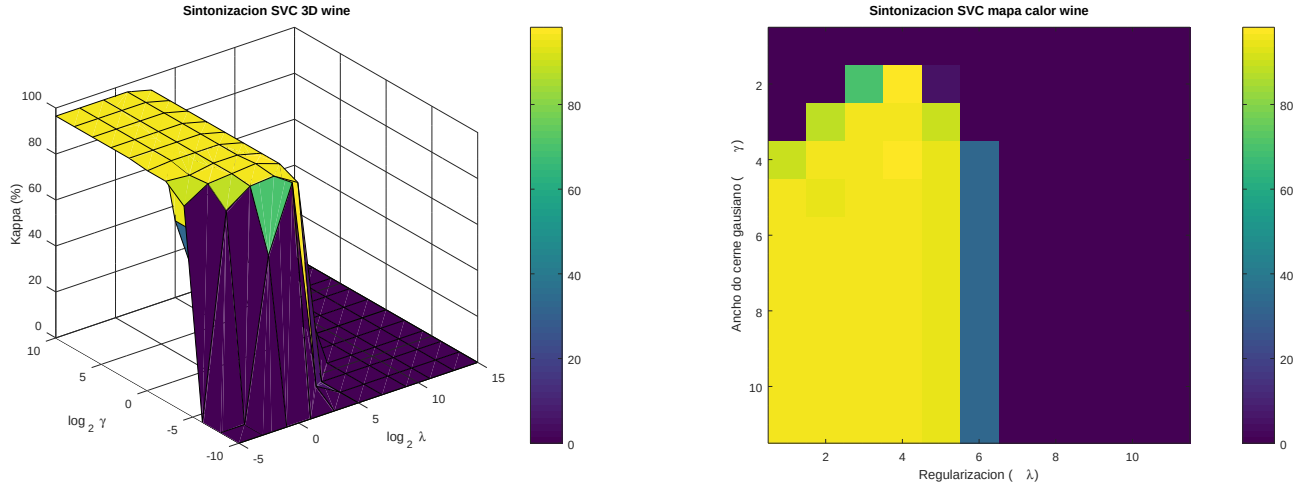

Figura 8: Diagrama 3D e mapa de calor co kappa en función de λ e γ na sintonización de SVC para o dataset `wine` creado polo programa `svc.m`.

2. Exercicios propostos

1. Implementa un clasificador Deep learning en R usando o paquete `deepnet`.

2. Instala o paquete `WekaDeeplearning4j` para executar unha rede profunda en Weka. Escribe un programa `deep.java` que use o obxecto `weka.Classifiers.functions.Dl4jMlpClassifier`

3. A Deep Learning toolbox de Matlab permite executar redes profundas nesta linguaxe.

Tema 6: Máquinas de vectores de soporte

1. Programas en Octave

1. Instala a libraría `Libsvm`. Vai ao directorio `libsvm-X.XX/matlab`, entra en Octave e executa `make`. Sal do Octave.

2. Escribe un programa `svc.m` que aprenda un problema de clasificación con validación cruzada 4-fold. Usa unha SVC con cerne gausiano sintonizando o hiper-parámetro de regularización (λ, nomeado C por Libsvm) e o inverso do ancho do cerne gausiano (nomeado γ por Libsvm). Xera unha gráfica coa sintonización similar ás xeradas nos temas anteriores. Xera tamén as gráficas da figura 8 co resultado da sintonización.

3. A partir de `svc.m`, crea o programa `svr.m` que execute o entrenamento, sintonización e teste dunha ε-SVR para un problema de regresión. Neste caso, debes usar a opción `-s 3` na variábel `opc` para este tipo de SVM. Os parámetros sintonizábeis son os mesmos que en SVC. Debes usar as medidas de calidade para regresión e xerar as gráficas correspondentes.

4. (Proposto) Amplía o programa `svr.m` para sintonizar tamén o parámetro ε, que por defecto vale 0.1, usando a opción `-p`. Xera a gráfica correspondente para esta sintonización.

2. Exercicios propostos

1. Modifica o programa `svc.m` para usar a aproximación multi-clase one-vs-all. Para isto, crea un vector de C SVCs de 2 clases, asignando cada patrón á clase que ten a SVC cunha meirande saída.

2. Implementa a SVM para clasificación e regresión usando os obxectos `SVC` e `SVR` do módulo `sklearn.svm`. Sintoniza os parámetros λ e γ.

3. Implementa a SVM en R coa función `ksvm` do paquete `kernlab`.

4. Executa a SVM en Weka usando o obxecto `weka.classifiers.functions.LibSVM`, que require a libraría Libsvm usada en Octave.

Tema 7: Combinacións de modelos

1. Programas en Matlab

1. Crea un programa chamado `cboost.m` que use a función `fitcensemble(x,y)` para executar un ensemble Boosting a problemas de clasificación usando validación cruzada 4-fold. Esta función usa o algoritmo LogitBoost para clasificación binaria e AdaboostM2 para multiclase.

```
x=load('hepatitis.data');
y=x(:,1);x(:,1)=[];
modelo=fitcensemble(x,y);  # entrenamento
z=predict(modelo,x);  # teste
```

2. Crea un programa `rboost.m` que use a función `fitrensemble` para executar Boosting (específicamente, LSBoost) en problemas de regresión.

3. Escribe unha función de Octave nomeada `ranking_friedman` que elabore o ranking de Friedman dos clasificadores `nn.m`, `lda.m`, `ctree.m`, `mlpc_id.m`, `svc.m` e `cboost.m` sobre os problemas de clasificación `hepatitis`, `wine`.

4. Executa os comandos de Octave necesarios para determinar cal dos clasificadores `svc.m` e `mlpc_id.m` é mellor e se a diferencia entre eles é estatísticamente significativa segundo o teste de rangos de signos de Wilcoxon.

2. Programas en Python

1. Escribe un programa `adaboostc.py` que use o obxecto `AdaboostClassifier` do módulo `sklearn.ensembles` para executar Adaboost en problemas de clasificación con validación cruzada 4-fold.

2. Crea un programa `rfc.py` que execute o Random Forest para problemas de clasificación usando o obxecto `RandomForestClassifier` do mesmo módulo.

3. Exercicios propostos

1. Implementa Adaboost para problemas de regresión en Python co obxecto `AdaBoostRegressor` do módulo `sklearn.ensembles`.
2. Implementa Adaboost en R usando a función `boosting` do paquete `adabag`.
3. Implementa Random forest en R usando a función `randomForest` do paquete co mesmo nome.
4. Weka proporciona os obxectos `weka.classifiers.meta.AdaboostM1` and `Bagging` para executar os ensembles Adaboost e Bagging e o obxecto `weka.classifiers.trees.RandomForest` para Random Forest.

Tema 8: Agrupamento

1. Programas en Octave

1. Escribe un programa `cluster.m` que use a función `kmeans` para agrupar patróns no problema de clasificación `hepatitis`. Mostra o erro de agrupamento.
2. Descarga as imaxes `oilred1.tif` e `lena.jpeg`. Escribe un programa `cluster_imaxe.m` que lea a imaxe coa función `imread` e a visualice coa función `imshow`. Convírtea a matriz e agrupa os píxeles coa función `kmeans`. Finalmente, convirte os índices dos grupos dos píxeles a matriz e visualiza esta matriz como imaxe. Proba con distintos valores de G (número de grupos).
3. Escribe un programa `gmm.m` que axuste un modelo de mestura gausiana (GMM) ao problema `wine` usando a función `fitgmdist` de Matlab.
4. Escribe o programa `gmm_2d.m` seguinte, que representa gráficamente (ver figura 9) as gausianas axustadas sobre un problema 2D artificial:

```
clear;clc
mu1 = [1 2];
Sigma1 = [2 0; 0 0.5];
mu2 = [-3 -5];
Sigma2 = [1 0;0 1];
rng(1); % For reproducibility
X = [mvnrnd(mu1,Sigma1,1000);mvnrnd(mu2,Sigma2,1000)];

GMModel = fitgmdist(X,2);

figure(1)
y = [zeros(1000,1);ones(1000,1)];
h = gscatter(X(:,1),X(:,2),y);
hold on
ezcontour(@(x1,x2)pdf(GMModel,[x1 x2]),get(gca,{'XLim','YLim'}))
title('{\bf Scatter Plot and Fitted Gaussian Mixture Contours}')
legend(h,'Grupo 1','Grupo 2')
hold off;grid on
print('-depsc','gmm_2d.eps')
```

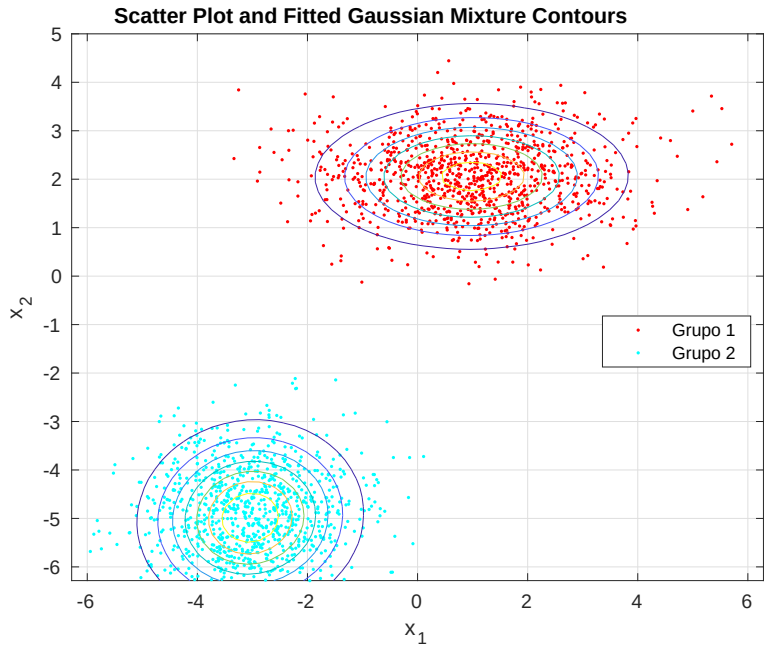

Figura 9: Grupos aprendidos por GMM en problema 2D artificial.

2. Programas en Python

1. Escribe un programa en Python nomeado `cluster.py` que agrupe os patróns do problema `hepatitis` usando os obxectos `KMeans`, `MiniBatchKMeans`, `AffinityPropagation`, `MeanShift`, `SpectralClustering`, `AgglomerativeClustering`, `DBSCAN`, `OPTICS` e `Birch`

2. No anterior programa `cluster.py`, co método `KMeans`, modifica o programa para que cree a matriz de confusión que obterías se asignases cada patrón á clase maioritaria entre os patróns asignados ao grupo ao que se asigna o patrón.

3. Exercicios propostos

1. Escribe un programa `cluster.r` en R que use a función `kmeans` do paquete `stats` para agrupar patróns no problema `wine`.

2. Escribe un programa `cluster.java` para Weka que use o algoritmo `weka.clusterers.SimpleKMeans` para aplicar agrupamento Kmeans ao problema de clasificación `hepatitis`.

Tema 9: Reducción da dimensionalidade

1. Programas en Octave

1. Escribe un programa en Octave nomeado `pca.m` que lea o dataset `hepatitis` e o clasifique usando 1NN sobre os datos completos, mostrando kappa e a matriz de confusión. Logo, executa PCA sobre este problema, e representa gráficamente as varianzas das compoñentes principais (figura 10). Logo, aplica a transformación PCA aos datos orixinais e repite a clasificación anterior, comparando os resultados obtidos.

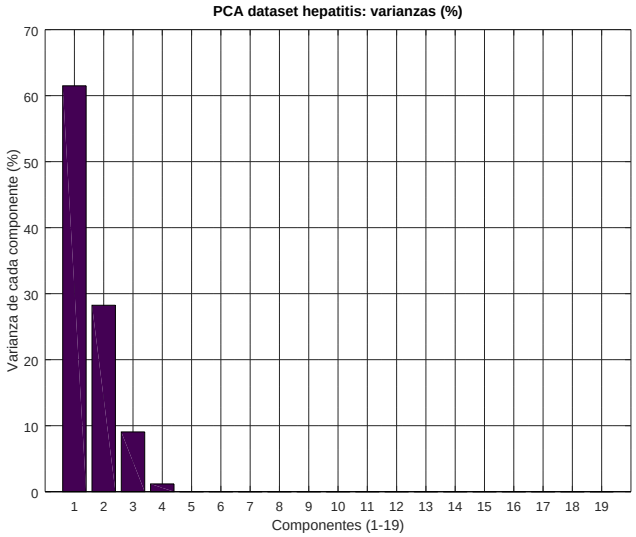

Figura 10: Varianzas das compoñentes principais para o problema `hepatitis`.

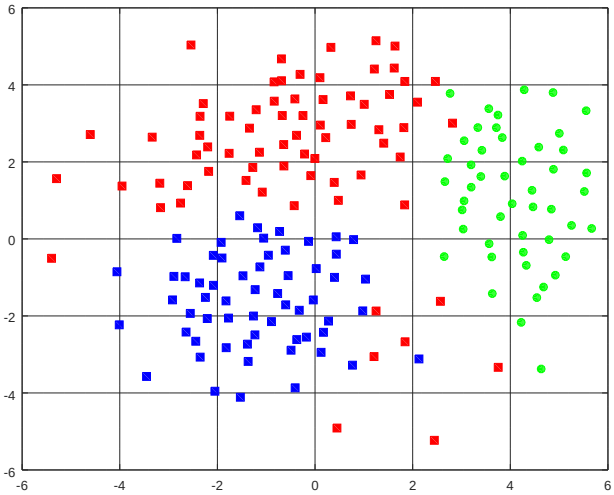

Figura 11: Problema `wine` proxectado a 2D usando a proxección de Sammon.

2. Executa o programa anterior co problema de clasificación `semeion.data`, da UCI (enlace), con 256 entradas, 1,593 patróns e 10 clases. Neste caso, a columna das clases é a última. Modifica o programa para que conte cantas compoñentes principais necesita para considerar o 95 % da varianza dos datos.

3. Compara os resultados cos datos orixinais e reducidos no problema de regresión `airfoil`.

4. Descarga a Dimensionality Reduction Toolbox de Laurens Van der Maaten dende este enlace. Escribe un programa chamado `redim_clasif.m` que use a función `sammon` desta toolbox para reducir a dimensionalidade do problema `wine`. Calcula o kappa obtido por un clasificador 5NN sobre os datos orixinais e proxectados en 2D, representando gráficamente estos últimos como na figura 11. Executa repetidamente esta proxección a m dimensións con m dende n a 2, e representa o kappa sobre os datos proxectados en función de m.

5. Modifica o programa anterior para que permita executar NCA para a reducción de dimensionalidade e a visualización supervisada do problema de clasificación `wine`.

6. Crea un programa `redim_reg.m` que aplique a función `sammon` ao problema de regresión `airfoil`.

2. Programas en Python

1. Crea un programa `pca.py` que usa o obxecto `PCA` do módulo `sklearn.decomposition` para aplicar PCA aos problemas de clasificación `hepatitis` e de regresión `airfoil`.

2. Crea un programa `selec_carac.py` que use os obxectos `VarianceThreshold` para seleccionar características baseándose en varianza, `SelectKBest` baseándose nun teste estatístico Chi-squared, baseándose en en modelo, p.ex. en clasificador `sklearn.linear_model.LassoCV` ou usando `SequentialFeatureSelector`, do módulo `sklearn.feature_selection`.

3. Exercicios propostos

1. Escribe un programa `pca.r` que aplique PCA usando a función `PCA` do paquete `FactoMineR` para reducir a dimensionalidade dos problemas anteriores e avalíe os resultados con e sen reducción da dimensionalidade.

2. Escribe un programa `pca.java` que aplique PCA ao problema de clasificación `semeion` anterior, usando para isto o obxecto `weka.attributeSelection.PrincipalComponents`.