International Master in Computer Vision

Fundamentals of machine learning for computer vision

Eva Cernadas

Contents

- **Machine learning theory (Dr. Jaime Cardoso)**
- **Linear regression and optimization (Dr. Jaime Cardoso)**
- **Clustering**
- **Model selection and evaluation**
- **Classical classification models**
- **Artificial neural networks**
- **Support vector machines (SVM)**
- **Ensembles: bagging, boosting and random forest**

Linear Discriminant Analysis

- **LDA: linear discriminant analysis**, 2 classes
- Projects to a one-dimensional space y=$\mathbf{w}^T\mathbf{x}$ that:

1) Maximizes the separation between the projected means of both classes: $m_i = \mathbf{w}^T \mathbf{m}_i$ with $i=1,2$ $\quad \boldsymbol{m}_i = \frac{1}{N_i} \sum_{y_j = i} \boldsymbol{x}_j$

2) Minimizes the separation among patterns within a class

- Separation measure within a class: sum of squared distance between pattern and class mean

$$s_i^2 = \sum_{y_j = i} (y_j - m_i)^2 = \sum_{y_j = i} (\boldsymbol{w}^T \boldsymbol{x}_j - \boldsymbol{w}^T \boldsymbol{m}_i)^2, i = 1,2$$

Pattern projection

Mean projection

LDA 2 classes

- Separation measure between classes: $(m_1 - m_2)^2$
- Fisher criterion: $$J(\boldsymbol{w}) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$
- Substituting m_i and s_i vs. **w**: $$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T \boldsymbol{S}_B \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S}_W \boldsymbol{w}}$$ Rayleigh quotient
- $\mathbf{S}_B$: covariance matrix *between* classes

$$\boldsymbol{S}_B = (\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^T \quad \boldsymbol{S}_W = \sum_{i=1}^{2} \sum_{y_j = i} (\boldsymbol{x}_j - \boldsymbol{m}_i)(\boldsymbol{x}_j - \boldsymbol{m}_i)^T$$

- $\mathbf{S}_W$: sum of *within* class covariance matrices

$\mathbf{S}_B$ and $\mathbf{S}_W$: squared matrices of size n

LDA 2 classes

- For maximizing J(**w**), require ∇J(**w**)=**0** , leading to:

$$\left(\boldsymbol{w}^T \boldsymbol{S}_B \boldsymbol{w}\right)\boldsymbol{S}_W \boldsymbol{w}=\left(\boldsymbol{w}^T \boldsymbol{S}_W \boldsymbol{w}\right)\boldsymbol{S}_B \boldsymbol{w}$$

- Since $\mathbf{S}_B\mathbf{w}=(\mathbf{m}_2-\mathbf{m}_1)(\mathbf{m}_2-\mathbf{m}_1)^T\mathbf{w}$, while $(\mathbf{m}_2-\mathbf{m}_1)^T\mathbf{w}$, $\mathbf{w}^T\mathbf{S}_B\mathbf{w}$ and $\mathbf{w}^T\mathbf{S}_W\mathbf{w}$ are scalars we achieve that, $\mathbf{S}_w\mathbf{w} \propto (\mathbf{m}_2-\mathbf{m}_1)$ and:

$$\mathbf{w} \propto \mathbf{S}_W^{-1}(\mathbf{m}_2-\mathbf{m}_1)$$

- Classification: $z(\mathbf{x})=sign(\mathbf{w}^T\mathbf{x}-b)$: b is the threshold
- Assuming that $\mathbf{S}_W^{-1}$ exists: $b=\dfrac{\boldsymbol{w}^T\left(\boldsymbol{m}_1+\boldsymbol{m}_2\right)}{2}$
- Projects to ℝ: y=-1 (resp. +1) for patterns of class 1 (resp. 2): $D=C$-1: dimensionality of mapped space

LDA multiclass (I)

- With $n>C>2$ classes, $\mathbf{y}_i=\mathbf{W}\mathbf{x}_i \in \mathbb{R}^D$, with $\mathbf{W}$ of degree Dxn
- Projects to a D-dimensional space, with $D=C-1<n$.
- The total covariance is:

$$\boldsymbol{S}_T=\sum_{i=1}^{C}\left(\boldsymbol{x}_i-\boldsymbol{m}\right)\left(\boldsymbol{x}_i-\boldsymbol{m}\right)^T \qquad \boldsymbol{m}=\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{x}_i=\frac{1}{N}\sum_{i=1}^{C}N_i\boldsymbol{m}_i$$

where N_i is the number of patterns of class i

- Considering that $\mathbf{S}_\mathrm{T}=\mathbf{S}_\mathrm{B}+\mathbf{S}_\mathrm{W}$

$$\boldsymbol{S}_B=\sum_{i=1}^{C}N_i\left(\boldsymbol{m}_i-\boldsymbol{m}\right)\left(\boldsymbol{m}_i-\boldsymbol{m}\right)^T \qquad \boldsymbol{S}_W=\sum_{i=1}^{C}\sum_{y_j=i}\left(\boldsymbol{x}_j-\boldsymbol{m}_i\right)\left(\boldsymbol{x}_j-\boldsymbol{m}_i\right)^T$$

LDA multiclass (II)

- In the $\mathbb{R}^D$ *space*, the matrices $\boldsymbol{S}_B^D$ and $\boldsymbol{S}_W^D$ are:

$$\boldsymbol{S}_B^D=\sum_{i=1}^{C} N_i(\boldsymbol{y}_i-\boldsymbol{\mu})(\boldsymbol{y}_i-\boldsymbol{\mu})^T \quad \boldsymbol{S}_W^D=\sum_{i=1}^{C}\sum_{y_j=i}(\boldsymbol{y}_j-\boldsymbol{\mu}_i)(\boldsymbol{y}_j-\boldsymbol{\mu}_i)^T \quad \boldsymbol{\mu}_i=\frac{1}{N_i}\sum_{y_j=i}\boldsymbol{y}_i$$

- Fisher criterion:

Covariance matrices
In the original n-dim. space

$$J(\boldsymbol{W})=Trace\left(\boldsymbol{S}_W^{-1}\boldsymbol{S}_B\right)$$

- The *D*x*n* matrix **W** maximizing *J*(**W**) contains the *D* principal eigenvectors (i.e. associated to the highest *D* eigenvalues) of $\mathbf{S}_W^{-1}\mathbf{S}_B$

LDA multiclass (III)

- *range*($\mathbf{S}_B$)$\leq C$-1, because $\mathbf{S}_B$ is the sum of C matrices of range 1 (each one is the outer product of 2 vectors)
- $\mathbf{S}_B$ *has a maximum of C-1* non-zero eigenvalues
- The projection to a (C-1)-dimensional space does not modify J($\mathbf{W}$)
- The (C-1)-dimensional space generated by the C-1 principal eigenvectors keeps much information of the original space.
- The LDA reduces the dimensionality from n to $D=C-1$

LDA in Octave for dimensionality reduction

- Projects to a *D*-dimensional space (*D*=*C*-1<n)
- With *C*=2 classes, *D* must be 1.
- If *n*<*D*, LDA can not be applied.
- *v*=eigenvector matrix; *l*=diagonal matrix with eigenvalues
- Descending order of eigenvalues to select the most important eigenvectors

```
D=input('D? ');
x=load('datos.dat');n=size(x,2);
if n<D; error('D>n'); end
y=x(:,n);x(:,n)=[];
C=numel(unique(y));
St=cov(x);Si=zeros(n);
for i=1:C
    j=find(y==i);ni=numel(j);
    Si=Si+ni*cov(x(j,:))/C;
end
Se=St-Si;m=min(C-1,D);
[v,l]=eig(Se,Si);
[l,i]=sort(diag(l),'descend');
w=v(:,l(1:m));
y=x*w;
```

LDA classification

- Test pattern **x** mapped to $\mathbb{R}^D$: **y=Wx**
- Bayes rule applied to **y**: the predicted class z is the one with the mean $\boldsymbol{\mu}_z$, scaled by its covariance $\boldsymbol{\Sigma}_z$, nearest to **y** in $\mathbb{R}^D$, weighted by its relative population N_z

$$z(\boldsymbol{x}) = \underset{k=1..C}{argmax} \left\{ \frac{N_k P_k(\boldsymbol{y})}{\sum_{l=1}^{C} N_l P_l(\boldsymbol{y})} \right\}$$

$$P_k(\boldsymbol{y}) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} \exp\left\{ \frac{-1}{2} (\boldsymbol{y} - \vec{\mu}_k) \Sigma_k^{-1} (\boldsymbol{y} - \vec{\mu}_k) \right\}, \boldsymbol{y} = \boldsymbol{W}\,\boldsymbol{x}$$

- Probabilistic nearest mean classifier in $\mathbb{R}^D$

UF2DC: ultra fast 2D classifier

- It is a the two-dimensional visual map classifier and regressor.
- Map high dimensional problems to 2D using LDA.
- The class contours in 2D is defined using the training patterns in the projected space.
- A test pattern **x** is assigned to the class associated in the projected space.

UF2DC: ultra fast 2D classifier

- *Published in:*

Neural Processing Letters (2023) 55:5377–5400
https://doi.org/10.1007/s11063-022-11090-3

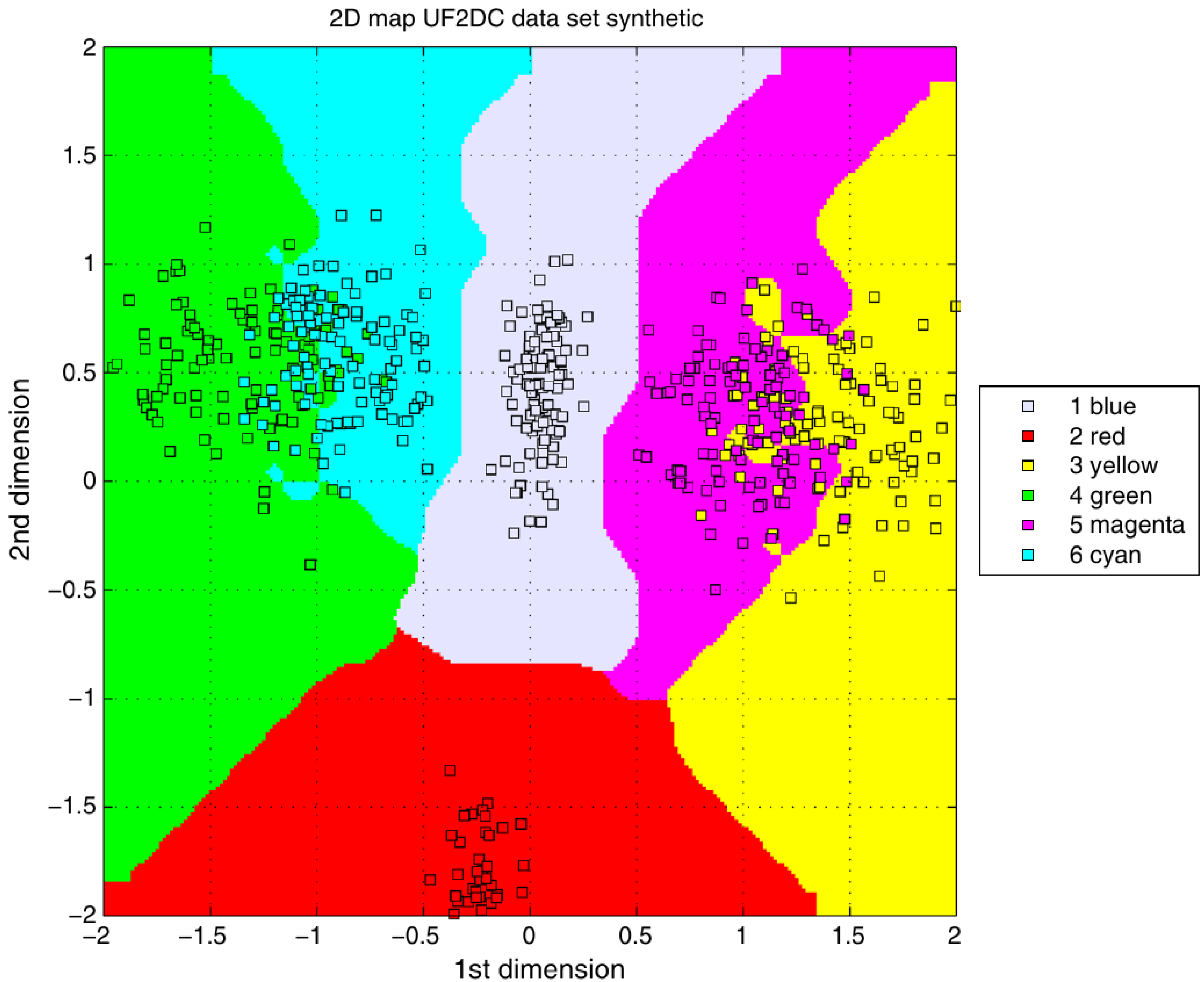
Ultra Fast Classification and Regression of High-Dimensional Problems Projected on 2D

Heba Alateyat[1] · Manuel Fernández-Delgado[1] · Eva Cernadas[1] · Senén Barro[1]

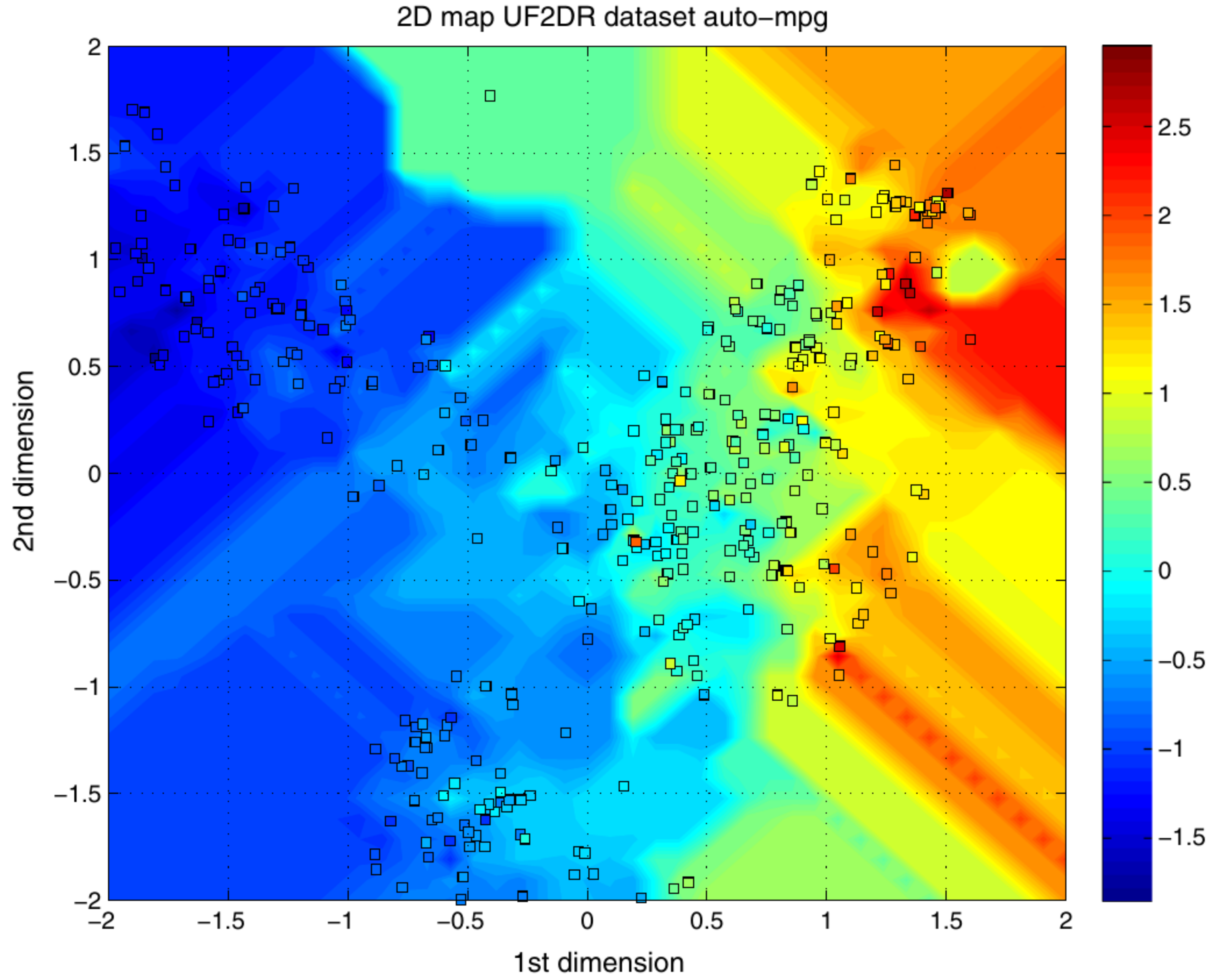
- *The code is available in:*

  https://persoal.citius.usc.es/manuel.fernandez.delgado/papers/uf2dcr/

UF2DC: ultra fast 2D classifier



UF2DR: ultra fast 2D regressor



LDA in Python (I)

- Implemented in **sklearn.discriminant_analysis**

sklearn.discriminant_analysis.LinearDiscriminantAnalysis

class `sklearn.discriminant_analysis.`**`LinearDiscriminantAnalysis`**(***, *solver='svd'*, *shrinkage=None*, *priors=None*, *n_components=None*, *store_covariance=False*, *tol=0.0001*) [source]

Linear Discriminant Analysis

A classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule.

LDA in Python (II)

- **sklearn.discriminant_analysis.LinearDiscriminantAnalysis** object.
- **Fit**() / **predict**() methods for training/testing.

```
from numpy import *
from sklearn.discriminant_analysis import *
from sklearn.metrics import *
tx=loadtxt('training_data.dat');ty=tx[:,0];tx=delete(tx,0,1)
sx=loadtxt('test_data.dat');sy=sx[:,0];sx=delete(sx,0,1)
model=LinearDiscriminantAnalysis().fit(tx,ty)
z=model.predict(sx)
acc=accuracy_score(y,z)
kappa=cohen_kappa_score(sy,z)
```

LDA classifier in Matlab

- Function **fitcdiscr**() for training.
- **predict**() for testing.

```
clear all
tx=load('training_data.dat');
ty=tx(:,1);tx=tx(:,2:end)
sx=load('test_data.dat');
sy=sx(:,);sx=sx(:,2:end)
model=fitcdiscr(tx,ty);
z=predict(model,sx)
kappa=calcula_kappa(sy,z)
```

LDA in R

- Implemented in package **MASS**, function **lda**

Examples

```
Iris <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
                   Sp = rep(c("s","c","v"), rep(50,3)))
train <- sample(1:150, 75)
table(Iris$Sp[train])
## your answer may differ
##  c  s  v
## 22 23 30
z <- lda(Sp ~ ., Iris, prior = c(1,1,1)/3, subset = train)
predict(z, Iris[-train, ])$class
##  [1] s s s s s s s s s s s s s s s s s s s s s s s s s s s c c c
## [31] c c c c c c c v c c c c v c c c c c c c c c c c v v v v v
## [61] v v v v v v v v v v v v v v v
(z1 <- update(z, . ~ . - Petal.W.))
```