# Aprendizagem Automática
## Aula Prática

## Modelos Lineares e Polinomiais

aplicados à

## Regressão

e à

## Classificação

G. Marques

# Modelos de Regressão

# Regressão Linear

- Objetivo:
  Analisar e inferir a relação entre uma variável dependente, $y$, e uma ou mais variáveis independentes, $x_1, x_2, \ldots$.
- Problema de aprendizagem supervisionada:
  Estimação do modelo baseada num conjunto de $N$ amostras de $y$ e num conjunto com os correspondentes $N$ vetores $\mathbf{x}$.
- Modelo linear:
  $$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d = \underbrace{\begin{bmatrix} w_0 & w_1 & \cdots & w_d \end{bmatrix}}_{\mathbf{w}^\top} \underbrace{\begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}}_{\mathbf{x}} = \mathbf{w}^\top \mathbf{x} = \mathbf{x}^\top \mathbf{w}$$
- Estimação dos parâmetros do modelo:
  Os valor do vetor de pesos, $\mathbf{w}$, é estimado através da minimização do erro quadrático médio.
  $$\mathcal{E} = \frac{1}{N} \sum_{n=1}^{N} \left( y[n] - \mathbf{w}^\top \mathbf{x}[n] \right)^2$$
  Solução obtida derivando o erro, igualando a zero, e resolvendo o sistema de equações resultante:
  $$\mathbf{w}_{\text{opt}} = \mathbf{R}_{\mathbf{x}}^{-1} \mathbf{r}_{\mathbf{x}y} = \left( \sum_n \mathbf{x}[n] \mathbf{x}[n]^\top \right)^{-1} \left( \sum_n y[n] \mathbf{x}[n] \right)$$

# Coeficiente $\mathrm{R}^2$

## Avaliação:

Uma maneira de determinar a qualidade do ajuste do modelo aos dados é através do do coeficiente $\mathrm{R}^2$ (ou coeficiente de determinação). Este coeficiente expressa a percentagem da variância da variável dependente que é predita pelas variáveis independentes.

- Média da variável dependente: $\mu_y = \frac{1}{N}\sum_{n=1}^{N} y[n]$
- Soma dos quadrados (proporcional à variância): $\mathrm{SQ_{tot}} = \sum_{n=1}^{N}(y[n] - \mu_y)^2$
- Soma dos quadrados explicados ($\hat{y}$ predição): $\mathrm{SQ_{exp}} = \sum_{n=1}^{N}(\hat{y}[n] - \mu_y)^2$
- Soma dos quadrados dos resíduos ($y - \hat{y}$): $\mathrm{SQ_{res}} = \sum_{n=1}^{N}(\hat{y}[n] - y[n])^2$
- Notar que $SQ_{tot} = SQ_{exp} + SQ_{res}$.
- $\mathrm{R}^2 = \frac{\mathrm{SQ_{exp}}}{\mathrm{SQ_{tot}}} = 1 - \frac{SQ_{res}}{SQ_{tot}}$

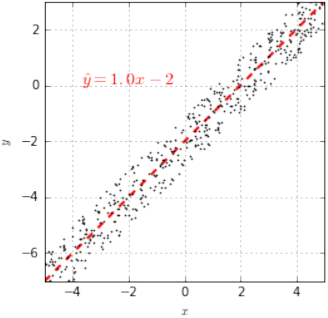
# Regressão Linear

## Exemplo: Pontos em torno de uma reta.

`RegressData001.p`

Objetivo: Determinar os pesos $w_0$ e $w_1$ do modelo de regressão linear e a validade da estimação.

- Modelo: $\hat{y} = w_1 x + w_0$
- Processo real: $y = x - 2 + \epsilon$
- $x$ - v.a. uniformemente distribuída entre $[-5, +5]$
- $\epsilon$ - ruído uniformemente distribuído entre $[-1, 1]$




- Nível de ruído em torno da reta depende da sua inclinação.

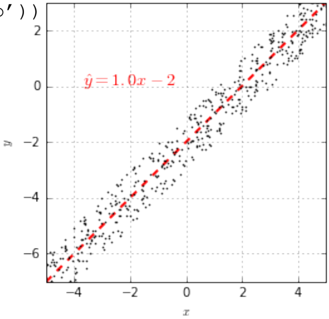
# Regressão Linear

## Exemplo: Pontos em torno de uma reta.

RegressData001.p

Objetivo: Determinar os pesos $w_0$ e $w_1$ do modelo de regressão linear e a validade da estimação.

```
# carregar dados - x e y(1 × 500)
>>>D=pickle.load(open('RegressData001.p'))
>>>(x,y)=(D['x'],D['y'])
# Construir X (2 × 500) - 2ª linha só com "1s"
>>>X=np.vstack((x,np.ones((1,500))))
>>>Rx=np.dot(X,X.T) # matrix 2 × 2
>>>rxy=np.dot(X,y.T) # vector 2 × 1
# Estimar pesos
>>>w=np.dot(np.linalg.pinv(Rx),rxy)
>>>print(w)
[[ 0.99728753]
 [-2.01391133]]
```



# Regressão Linear

## Exemplo: Pontos em torno de uma reta.

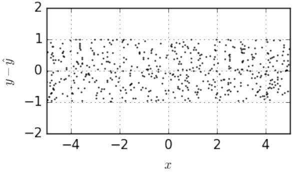
`RegressData001.p`

Objetivo: Determinar os pesos $w_0$ e $w_1$ do modelo de regressão linear e a validade da estimação.

- Visualizar o erro

  # calcular yh: estimativa de y ($1 \times 500$)
  ```
  >>>yh=np.dot((w.T,X))
  ```
  # visualizar erro
  ```
  >>>plt.plot(x,y-yh,'.k')
  ```

- Erro com distribuição aleatória uniforme.



# Regressão Linear

## Exemplo: Pontos em torno de uma reta.

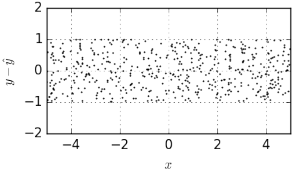
`RegressData001.p`

Objetivo: Determinar os pesos $w_0$ e $w_1$ do modelo de regressão linear e a validade da estimação.

- Visualizar o erro

```
# calcular yh: estimativa de y (1 × 500)
>>>yh=np.dot((w.T,X))
# visualizar erro
>>>plt.plot(x1,y-yh,'.k')
```

- Erro com distribuição aleatória uniforme.




- Cálculo do coeficiente $R^2$

```
>>> my=np.mean(y)  # média de y
>>> SQtot=np.sum((y-my)**2)# soma dos quadrados de y
>>> SQres=np.sum((y-yh)**2)# potência do erro
>>> R2=1.0-SQres/SQtot
0.96545...
```

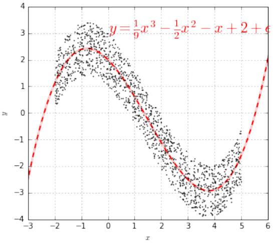
# Regressão Linear

## Exemplo: Pontos em torno de uma curva.

`RegressData002.p`

Objetivo: Determinar os pesos $w_0$ e $w_1$ do modelo de regressão linear e a validade da estimação.

- Modelo: $\hat{y} = w_1 x + w_0$
- Processo real: $y = \frac{1}{9}x^3 - \frac{1}{2}x^2 - x + 2 + \epsilon$
- $x$ - v.a. uniformemente distribuída entre $[-2, +5]$
- $\epsilon$ - ruído uniformemente distribuído entre $[-1, 1]$



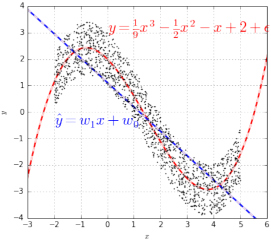
# Regressão Linear

## Exemplo: Pontos em torno de uma curva.

`RegressData002.p`

Objetivo: Determinar os pesos $w_0$ e $w_1$ do modelo de regressão linear e a validade da estimação.

```
# carregar dados - x e y(1 × 1500)
>>>D=pickle.load(open('RegressData002.p'))
>>>(x,y)=(D['x'],D['y'])
# Construir X (2 × 1500) - 2ª linha só com "1s"
>>>X=np.vstack((x,np.ones((1,1500))))
>>>Rx=np.dot(X,X.T) # matriz 2 × 2
>>>rxy=np.dot(X,y.T) # vetor 2 × 1
# Estimar pesos
>>>w=np.dot(np.linalg.pinv(Rx),rxy)
[[-0.91987301]
 [ 1.11538364]]
```



# Regressão Linear

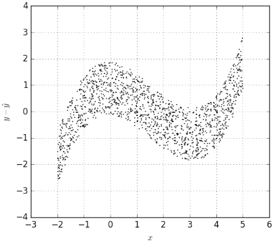
## Exemplo: Pontos em torno de uma curva.

`RegressData002.p`

Objetivo: Determinar os pesos $w_0$ e $w_1$ do modelo de regressão linear e a validade da estimação.

- Visualizar erro e calcular $R^2$

```
>>>yh=np.dot(w.T,X)
>>>plt.plot(x,y-yh,'.')
# Calcular o coeficiente R²
>>>my=np.mean(y
>>>SQtot=np.sum((y-my)**2)
>>>SQres=np.sum((y-yh)**2)
>>>R2=1.0-SQres/SQtot
R2=0.800
```




- Este valor de $R^2$ é bastante elevado, visto que o modelo não descreve adequadamente o comportamento dos dados. Ao visualizar o erro vemos que este não exibe um comportamento aleatório, sinal de que a estimação é fraca.

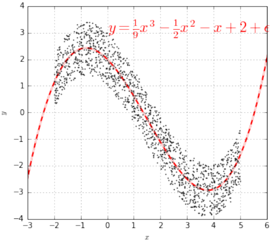
# Regressão Polinomial

## Exemplo: Pontos em torno de uma curva.

`RegressData002.p`

Objetivo: Determinar parâmetros do modelo de regressão polinomial de 3ª ordem e confirmar a validade da estimação.

- Modelo: $\hat{y} = w_3 x_1^3 + w_2 x_1^2 + w_1 x_1 + w_0$
- Processo real: $y = \frac{1}{9}x^3 - \frac{1}{2}x^2 - x + 2 + \epsilon$
- $x$ - v.a. uniformemente distribuída entre $[-2, +5]$
- $\epsilon$ - ruído uniformemente distribuído entre $[-1, 1]$



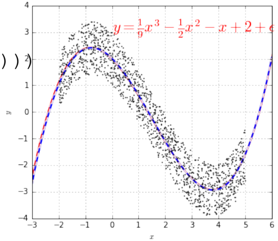
# Regressão Polinomial

## Exemplo: Pontos em torno de uma curva.

`RegressData002.p`

Objetivo: Determinar parâmetros do modelo de regressão polinomial de 3ª ordem e confirmar a validade da estimação.

```
# Construir matriz X (4 × 1500) 4ª linha só com "1s"
>>>X=np.vstack((x**3,x**2,x,np.ones((1,1500))))
>>>Rx=np.dot(X,X.T) # matriz 4 × 4
>>>rxy=np.dot(X,yh.T) # vetor 4 × 1
# Estimar pesos
>>>w=np.dot(np.linalg.pinv(Rx),rxy)
[[ 0.11197625],
 [-0.50684433],
 [-0.99269456],
 [ 2.01606518]]
```



# Regressão Polinomial

## Exemplo: Pontos em torno de uma curva.

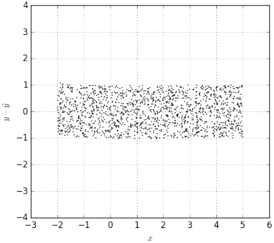
`RegressData002.p`

Objetivo: Determinar parâmetros do modelo de regressão polinomial de 3ª ordem e confirmar a validade da estimação.

- Visualizar o erro

```
# calcular yh: estimativa de y (1 × 500)
>>>yh=np.dot((w.T,X))
# visualizar erro
>>>plt.plot(x1,y-yh,'.k')
```

- Erro com distribuição aleatória uniforme.




- Cálculo do coeficiente $R^2$

```
>>> my=np.mean(y)  # média de y
>>> SQtot=np.sum((y-my)**2)# soma dos quadrados de y
>>> SQres=np.sum((y-yh)**2)# potência do erro
>>> R2=1.0-SQres/SQtot
0.92558...
```

# Regressão

## Regressão linear com `sklearn`

Dados reais: preço de casas na cidade de Boston nos anos 80s (dados no sub-módulo `datasets`).

```
>>> BH=load_boston()
>>> print(BH.DESC)
```

```
Boston House Prices dataset
Data Set Characteristics:
    :Number of Instances: 506
    :Number of Attributes: 13 numeric/categorical predictive
    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Creator: Harrison, D. and Rubinfeld, D.L.
```

# Regressão

## Regressão linear com `sklearn`

Dados reais: preço de casas na cidade de Boston nos anos 80s (dados no sub-módulo `datasets`).

Regressão linear está implementada no sub-módulo `linear_model`.

```
>>> from sklearn.linear_model import LinearRegression
>>> Xtrain,Xtest,ytrain,ytest=train_test_split(BH.data,BH.target)
>>> linReg=LinearRegression().fit(Xtrain,ytrain)
```

Os coeficientes da regressão (o vetor **w** - menos o peso $w_0$) estão guardados no atributo `coef_` e o peso $w_0$ está guardado em `intercept_`.

```
>>> print(lingReg.intercept_)
37.99
>>> print(lingReg.coef_)
[ -1.19858618e-01, 4.44233009e-02, 1.18612465e-02, 2.51295058e+00, -1.62710374e+01,
3.84909910e+00, -9.85471557e-03, -1.50002715e+00, 2.41507916e-01, -1.10671867e-02,
-1.01897720e+00, 6.95273216e-03, -4.88110587e-01])
```

Calculado "à mão":

```
>>> Xtrain2=np.vstack((np.ones(Xtrain.shape[0]),Xtrain.T))
>>> Rxi=np.linalg.pinv(np.dot(Xtrain2.T,Xtrain2))
>>> rxy=np.dot(Xtrain2.T,ytrain.T)
>>> w=np.dot(Rxi,rxy)
[ 3.79925928e+01 -1.19858618e-01 4.44233009e-02 1.18612465e-02 2.51295058e+00 -1.62710374e+01
3.84909910e+00 -9.85471557e-03 -1.50002715e+00 2.41507916e-01 -1.10671867e-02 -1.01897720e+00
6.95273216e-03 -4.88110587e-01]
```

# Regressão

## Regressão linear com `sklearn`

Dados reais: preço de casas na cidade de Boston nos anos 80s (dados no sub-módulo `datasets`).

Regressão linear está implementada no sub-módulo `linear_model`.

```
>>> from sklearn.linear_model import LinearRegression
>>> Xtrain,Xtest,ytrain,ytest=train_test_split(BH.data,BH.target)
>>> linReg=LinearRegression().fit(Xtrain,ytrain)
```

O cálculo do coeficiente $R^2$ é feito com o método `score` do regressor.

```
>>> print('coeficiente R2 (treino):  %f'%linReg.score(Xtrain,ytrain)))
>>> print('coeficiente R2 (teste):  %f'%linReg.score(Xtest,ytest)))
coeficiente R2 (treino):  0.764456
coeficiente R2 (teste) :  0.673528
```

Calculado "à mão":

```
>>> my=np.mean(ytrain))
>>> SQtot=np.sum((ytrain-my)**2)
>>> yh=linReg.predict(Xtrain)
>>> SQres=np.sum((ytrain-yh)**2)
>>> R2=1.0-SQres/SQtot
coeficiente R2 (treino):  0.764456
```

# Regressão

## Regressão polinomial com `sklearn`

Dados reais: preço de casas na cidade de Boston nos anos 80s (dados no sub-módulo `datasets`).

A função `LinearRegresion` pode implementar igualmente uma regressão polinomial. Basta transformar a matriz das variáveis independentes de modo a esta conter uma combinação polinomial das características. Esta transformação pode ser feita através da função `PolynomialFeatures` do sub-módulo `preprocessing`.

- A função `PolynomialFeatures` devolve a combinação polinomial das características, incluindo a coordenada homogénea (coluna de $1_s$). É necessário remover esta coordenada antes de fazer a regressão - a função `LinearRegression` acrescenta a coordenada homogénea internamente.
- A dimensão dos dados é 13. Isto faz com que os termos polinomiais sejam um total de 105, incluindo a coordenada homogénea.

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly=PolynomialFeatures(2).fit(Xtrain)
>>> Xtrain2=poly.transform(Xtrain)[:,1:]
>>> Xtest2=poly.transform(Xtest)[:,1:]
>>> polyReg=LinearRegression().fit(Xtrain2,ytrain)
>>> print('coeficiente R2 (treino):  %f'%polyReg.score(Xtrain2,ytrain))
>>> print('coeficiente R2 (teste):  %f'%polyReg.score(Xtest2,ytest))
coeficiente R2 (treino):  0.952154
coeficiente R2 (teste) :  0.644542
```

# Regressão

## Regularização com `sklearn`

Dados reais: preço de casas na cidade de Boston nos anos 80s (dados no sub-módulo `datasets`).

A regressão polinomial pode facilmente entrar em sobre aprendizagem, particularmente em situações em que a dimensão dos dados é elevada e o número de exemplos limitado, como é o caso destes dados. O sub-módulo `linear_model` tem dois métodos que implementam a regressão linear com regularização dos pesos: `Ridge` e `Lasso`.

**Ridge:** A soma do quadrado dos pesos **w** pesada por um termo $\alpha$ é adicionada à função do erro.

$$\mathcal{E}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(y[n] - \mathbf{w}^\top \mathbf{x}[n]\right)^2 + \alpha \sum_{i=0}^{d} w_i^2$$

**Lasso:** A soma do valor absoluto dos pesos **w** pesada por um termo $\alpha$ é adicionada à função do erro.

$$\mathcal{E}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(y[n] - \mathbf{w}^\top \mathbf{x}[n]\right)^2 + \alpha \sum_{i=0}^{d} |w_i|$$

# Regressão

## Regularização com sklearn

Dados reais: preço de casas na cidade de Boston nos anos 80s (dados no sub-módulo datasets).

A regressão polinomial pode facilmente entrar em sobre aprendizagem, particularmente em situações em que a dimensão dos dados é elevada e o número de exemplos limitado, como é o caso destes dados. O sub-módulo linear_model tem dois métodos que implementam a regressão linear com regularização dos pesos: Ridge e Lasso.

### Ridge

```
>>> from sklearn.linear_model import Ridge
>>> poly=PolinomialFeatures(2).fit(Xtrain)
>>> Xtrain2=poly.transform(Xtrain)[:,1:]
>>> Xtest2=poly.transform(Xtest)[:,1:]
>>> ridge=Ridge(alpha=1.0).fit(Xtrain2,ytrain)
>>> print('coeficiente R2 (treino):  %f'%ridge.score(Xtrain2,ytrain)))
>>> print('coeficiente R2 (teste):  %f'%ridge.score(Xtest2,ytest)))
coeficiente R2 (treino):  0.948191
coeficiente R2 (teste) :  0.635045
>>> ridge=Ridge(alpha=100.0).fit(Xtrain2,ytrain)
>>> print('coeficiente R2 (treino):  %f'%ridge.score(Xtrain2,ytrain)))
>>> print('coeficiente R2 (teste):  %f'%ridge.score(Xtest2,ytest)))
coeficiente R2 (treino):  0.931567
coeficiente R2 (teste) :  0.773319
```

# Regressão

## Regularização com sklearn

Dados reais: preço de casas na cidade de Boston nos anos 80s (dados no sub-módulo `datasets`).

A regressão polinomial pode facilmente entrar em sobre aprendizagem, particularmente em situações em que a dimensão dos dados é elevada e o número de exemplos limitado, como é o caso destes dados. O sub-módulo `linear_model` tem dois métodos que implementam a regressão linear com regularização dos pesos: `Ridge` e `Lasso`.

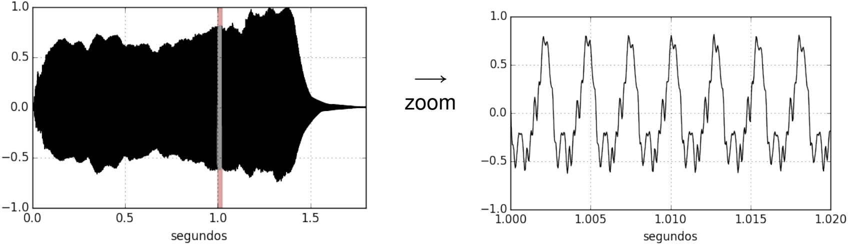
### Lasso

```
>>> from sklearn.linear_model import Lasso
>>> lasso=Lasso(alpha=1.0,max_iter=1000).fit(Xtrain2,ytrain)
>>> print('coeficiente R2 (treino):  %f'%lasso.score(Xtrain2,ytrain)))
>>> print('coeficiente R2 (teste):  %f'%lasso.score(Xtest2,ytest)))
coeficiente R2 (treino):  0.907694
coeficiente R2 (teste) :  0.749142
>>> lasso=Lasso(alpha=0.1).fit(Xtrain2,ytrain)
>>> print('coeficiente R2 (treino):  %f'%lasso.score(Xtrain2,ytrain)))
>>> print('coeficiente R2 (teste):  %f'%lasso.score(Xtest2,ytest)))
coeficiente R2 (treino):  0.923551
coeficiente R2 (teste) :  0.786641
```

# Regressão Linear

## Sinais áudio de instrumentos musicais

Neste exemplo, analisamos o sinal de áudio correspondente a uma nota musical tocada num violino (ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`). Este sinal faz parte de uma base de dados de notas musicais tocadas por variados instrumentos, que foi desenvolvida pelo grupo de investigação Electronic Music Studios da Universidade de Iowa.
Tipicamente, notas produzidas por instrumento musicais são sinais que exibem uma alta periodicidade (há exceções como é o caso dos instrumentos de percussão).

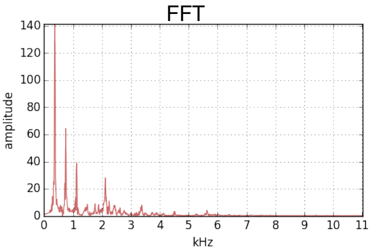


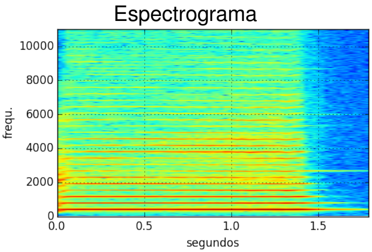

Representações no domínio do tempo

# Regressão Linear

## Sinais áudio de instrumentos musicais

Neste exemplo, analisamos o sinal de áudio correspondente a uma nota musical tocada num violino (ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`). Este sinal faz parte de uma base de dados de notas musicais tocadas por variados instrumentos, que foi desenvolvida pelo grupo de investigação Electronic Music Studios da Universidade de Iowa.
Tipicamente, notas produzidas por instrumento musicais são sinais que exibem uma alta periodicidade (há exceções como é o caso dos instrumentos de percussão).




Representações no domínio da frequência

# Regressão Linear

## Sinais áudio de instrumentos musicais

- Objectivo: Dado o sinal de áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`, prever o valor do sinal no instante $n$ baseado numa regressão linear das $p$ amostras anteriores.

$$\begin{aligned}\hat{x}[n] &= \sum_{i=1}^{p} w_i x[n-i] = \mathbf{w}^\top \mathbf{x} = \begin{bmatrix} w_1 & w_2 & \dots & w_p \end{bmatrix} \begin{bmatrix} x[n-1] \\ x[n-2] \\ \vdots \\ x[n-p] \end{bmatrix} \\ x[n] &= \hat{x}[n] + \epsilon[n] = \mathbf{w}^\top \mathbf{x} + \epsilon[n] \\ \epsilon[n] &\quad \text{ruído (supostamente branco e gaussiano)}\end{aligned}$$

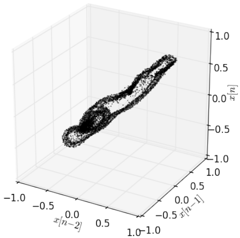
- Nota: Esta é uma técnica denominada Codificação Linear Preditiva (LPC - do inglês Linear Predictive Coding), uma metodologia que aproxima um dado som através da resposta de um filtro IIR (excitado com um trem diracs ou com ruído).
  - Função de transferência: $H(z) = \dfrac{1}{1 - \sum_{i=1}^{p} w_i z^{-i}}$
  - Muito usado em processamento digital de fala

# Regressão Linear

## Sinais áudio de instrumentos musicais

- Objectivo: Dado o sinal de áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`, prever o valor do sinal no instante $n$ baseado numa regressão linear das $p$ amostras anteriores.
- Confirmar, através de visualização gráfica, se o sinal, $x[n]$ presta-se a ser modelado por uma regressão linear. Agrupar amostras todas as 3 amostras consecutivas de $x[n]$ e visualizar.

```
#módulo de leitura de .wav
>>> import scipy.io.wavfile as wav
>>> fs,x=wav.read('Violin.arco.ff.sulG.Gb4.mono.wav')
>>> x=x.astype('float') #converter para float
>>> x=x/2.0**15 #amplitude [-1,+1]
>>> x=x[1000:6001] #ver 5000 amostras
#previamente definido ax eixo 3D
>>> ax.plot(x[0:-2],x[1:-1],x[2:],'.k')
```

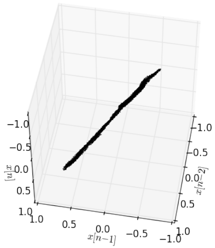


# Regressão Linear

## Sinais áudio de instrumentos musicais

- Objectivo: Dado o sinal de áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`, prever o valor do sinal no instante *n* baseado numa regressão linear das *p* amostras anteriores.
- Confirmar, através de visualização gráfica, se o sinal, $x[n]$ presta-se a ser modelado por uma regressão linear. Agrupar amostras todas as 3 amostras consecutivas de $x[n]$ e visualizar.

```
#módulo de leitura de .wav
>>> import scipy.io.wavfile as wav
>>> fs,x=wav.read('Violin.arco.ff.sulG.Gb4.mono.wav')
>>> x=x.astype('float') #converter para float
>>> x=x/2.0**15 #amplitude [-1,+1]
>>> x=x[1000:6001] #ver 5000 amostras
#previamente definido ax eixo 3D
>>> ax.plot(x[0:-2],x[1:-1],x[2:],'.k')
```



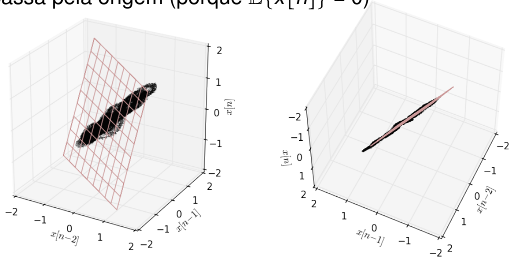
# Regressão Linear

## Sinais áudio de instrumentos musicais

- Objectivo: Dado o sinal de áudio, prever o valor do sinal no instante $n$ baseado numa regressão linear das duas amostras anteriores.

$$
\begin{aligned}
\hat{x}[n] &= w_1 x[n-1] + w_2 x[n-1] = \mathbf{w}^\top \mathbf{x} = \begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} x[n-1] \\ x[n-2] \end{bmatrix} \\
x[n] &= \hat{x}[n] + \epsilon[n] = \mathbf{w}^\top \mathbf{x} + \epsilon[n] \\
\\
\epsilon[n] &\quad \text{ruído (supostamente branco e gaussiano)}
\end{aligned}
$$

  - $x[n]$ sinal áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`
  - Neste caso não é necessário incluir o termo $w_0$
  - Plano passa pela origem (porque $\mathbb{E}\{x[n]\} = 0$)

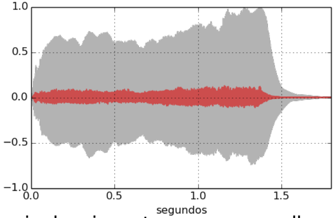


# Regressão Linear

## Sinais áudio de instrumentos musicais (`LinMod_Music.py`)

- Objectivo: Dado o sinal de áudio, prever o valor do sinal no instante $n$ baseado numa regressão linear das 20 amostras anteriores.
  - $x[n]$ sinal áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`
  - Visualizar erro de estimação
  - Analisar o modelo estimado como uma filtragem auto-regressiva (filtro IIR)
  - Ouvir sinal sintetizado com o filtro IIR

```
# construir matriz X (20 × N)
>>>X=np.vstack((x[19:-1],x[18:-2],...\
...  x[1:-19],x[0:-20]))
# matriz Y (1 × N)
>>>Y=x[np.newaxis,20:]
>>>Rx=np.dot(X,X.T) # matrix 20 × 20
>>>rxy=np.dot(X,Y.T) # vector 20 × 1
>>>w=np.dot(np.linalg.inv(Rx),rxy)
>>>Erro=Y-np.dot(w.T,X)
```




sinal a cinzento, erro a vermelho

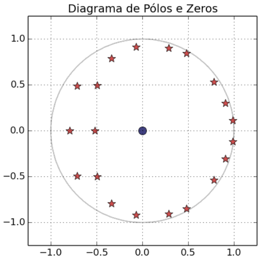
# Regressão Linear

## Sinais áudio de instrumentos musicais (`LinMod_Music.py`)

- Objectivo: Dado o sinal de áudio, prever o valor do sinal no instante $n$ baseado numa regressão linear das 20 amostras anteriores.
  - $x[n]$ sinal áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`
  - Visualizar erro de estimação
  - Analisar o modelo estimado como uma filtragem auto-regressiva (filtro IIR)
  - Ouvir sinal sintetizado com o filtro IIR

### Analisar regressão como uma filtragem

- Equação às diferenças:
  $x[n] = w_1 x[n-1] + \ldots + w_{20} x[n-20] + \epsilon[n]$
- Função de transferência:
  $$H(z) = \frac{1}{1 - \sum_{i=1}^{20} w_i z^{-i}}$$



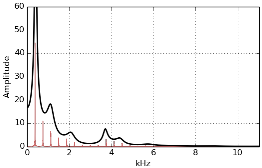
# Regressão Linear

## Sinais áudio de instrumentos musicais (`LinMod_Music.py`)

- Objectivo: Dado o sinal de áudio, prever o valor do sinal no instante $n$ baseado numa regressão linear das 20 amostras anteriores.
  - $x[n]$ sinal áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`
  - Visualizar erro de estimação
  - Analisar o modelo estimado como uma filtragem auto-regressiva (filtro IIR)
  - Ouvir sinal sintetizado com o filtro IIR

### Analisar regressão como uma filtragem

- Espectro do sinal (a vermelho) obtido com a função `np.fft.fft()`
- Espectro do filtro (a preto) obtido com a função `scipy.signal.freqz()`




Espectro do filtro acompanha os picos do espectro do sinal
A resposta em frequência deste tipo de filtros é denominada a envolvente espectral do sinal.

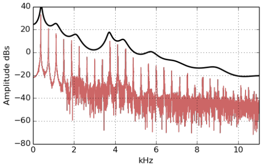
# Regressão Linear

## Sinais áudio de instrumentos musicais (`LinMod_Music.py`)

- Objectivo: Dado o sinal de áudio, prever o valor do sinal no instante $n$ baseado numa regressão linear das 20 amostras anteriores.
  - $x[n]$ sinal áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`
  - Visualizar erro de estimação
  - Analisar o modelo estimado como uma filtragem auto-regressiva (filtro IIR)
  - Ouvir sinal sintetizado com o filtro IIR

### Analisar regressão como uma filtragem

- Espectro do sinal (a vermelho) obtido com a função `np.fft.fft()`
- Espectro do filtro (a preto) obtido com a função `scipy.signal.freqz()`




Espectro do filtro acompanha os picos do espectro do sinal
A resposta em frequência deste tipo de filtros é denominada a envolvente espectral do sinal.

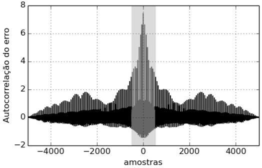
# Regressão Linear

## Sinais áudio de instrumentos musicais (`LinMod_Music.py`)

- Objectivo: Dado o sinal de áudio, prever o valor do sinal no instante $n$ baseado numa regressão linear das 20 amostras anteriores.
  - $x[n]$ sinal áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`
  - Visualizar erro de estimação
  - Analisar o modelo estimado como uma filtragem auto-regressiva (filtro IIR)
  - Ouvir sinal sintetizado com o filtro IIR

### Analisar regressão como uma filtragem

- A resposta impulsiva do filtro sintetiza o som do sinal mas só dura alguns milissegundos.
- É necessário obter a resposta impulsiva periódica (usar um trem de diracs como sinal de entrada).
- Para saber o espaçamento entre impulsos visualizar a **função de autocorrelação** do erro.



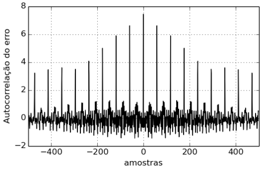
# Regressão Linear

## Sinais áudio de instrumentos musicais (`LinMod_Music.py`)

- Objectivo: Dado o sinal de áudio, prever o valor do sinal no instante $n$ baseado numa regressão linear das 20 amostras anteriores.
  - $x[n]$ sinal áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`
  - Visualizar erro de estimação
  - Analisar o modelo estimado como uma filtragem auto-regressiva (filtro IIR)
  - Ouvir sinal sintetizado com o filtro IIR

### Analisar regressão como uma filtragem

- A resposta impulsiva do filtro sintetiza o som do sinal mas só dura alguns milissegundos.
- É necessário obter a resposta impulsiva periódica (usar um trem de diracs como sinal de entrada).
- Para saber o espaçamento entre impulsos visualizar a **função de autocorrelação** do erro.



# Regressão Linear

## Sinais áudio de instrumentos musicais (`LinMod_Music.py`)

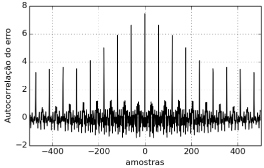
- Objectivo: Dado o sinal de áudio, prever o valor do sinal no instante $n$ baseado numa regressão linear das 20 amostras anteriores.
  - $x[n]$ sinal áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`
  - Visualizar erro de estimação
  - Analisar o modelo estimado como uma filtragem auto-regressiva (filtro IIR)
  - Ouvir sinal sintetizado com o filtro IIR

### Analisar regressão como uma filtragem

- Função de autocorrelação, $R_x[k]$ dum sinal $x[n]$:

$$R_x[k] = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} x[n]x[n-k]$$

- $R_x[k]$ é uma medida de semelhança entre valores de $x[n]$ separados de $k$ amostras
- Obtido com `sp.signal.convolve()`



# Regressão Linear

## Sinais áudio de instrumentos musicais (`LinMod_Music.py`)

- Objectivo: Dado o sinal de áudio, prever o valor do sinal no instante $n$ baseado numa regressão linear das 20 amostras anteriores.
  - $x[n]$ sinal áudio do ficheiro `Violin.arco.ff.sulG.Gb4.mono.wav`
  - Visualizar erro de estimação
  - Analisar o modelo estimado como uma filtragem auto-regressiva (filtro IIR)
  - Ouvir sinal sintetizado com o filtro IIR

### Sintetizar o sinal através duma filtragem

```
# construir sinal de entrada (N=nº amostras)
>>>trDirac=np.zeros(N)
# inspeção visual: espaçamento 59 amostras
>>>trDirac[np.arange(0,x.shape[0],59)]=1.0
# sintetizar sinal: a=coeficentes do filtro
# a=[1 -w_1 -w_2 ··· -w_p]
>>>xSint=sg.lfilter(1,a[:,0],trDirac)
```

# Modelos de Classificação

# Discriminantes Lineares

## Exemplo: Discriminantes lineares (2 classes)

- Objectivo: Separar imagens de "$0_s$" e de "$1_s$"

```
>>> D=pickle.load(open('MNISTsmall.p','rb'))
>>> X0,X1=D['X'][:,:1000],D['X'][:,1500:2500] # np.arrays de 784×1000
>>> X=np.vstack(np.ones(2000),np.hstack((X0,X1))) #matriz de 785×2000
>>> Y=np.hstack(-np.ones(1000),np.ones(1000)) # matriz de 1×2000
```

- Projecção linear calculada com dados de treino (2000 imagens total)

$$\hat{y} = \mathbf{w}^{\mathsf{T}}\mathbf{x} \text{ com } \mathbf{w} = \mathbf{R}_{\mathbf{x}}^{-1}\mathbf{r}_{\mathbf{x}y}$$

```
>>> Rx=np.dot(X,X.T)
>>> rxy=np.dot(X,Y.T)
```

Transformação

```
>>> w=np.dot(np.linalg.inv(Rx),rxy)
LinAlgError:Singular Matrix
```

- Problema: $\mathbf{R}_{\mathbf{x}}$ é uma matriz singular (o inverso dá erro no Python)

# Discriminantes Lineares

## Exemplo: Discriminantes lineares (2 classes)

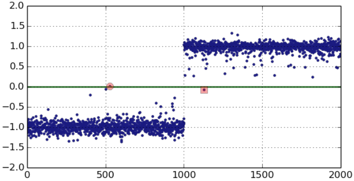
- Solução 1: pré-processar dados com PCA
  (Guardar componentes com valores próprios >> 0 )
- Solução 2: Usar a pseudo-inversa de Moore Penrose para calcular $\mathbf{R}_\mathbf{x}^{-1}$
  A pseudo-inversa de Moore Penrose é uma generalização do inverso de uma matriz.
  Este método pode ser aplicado a matrizes singulares.

```
>>> Rxi=np.linalg.pinv(Rx)
>>> w=np.dot(Rxi,rxy)
```

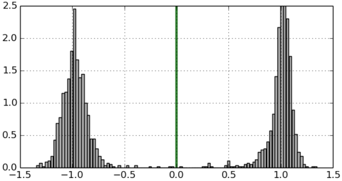
# Discriminantes Lineares

## Exemplo: Discriminantes lineares (2 classes)

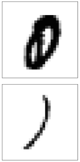
- Resultados – dados de treino




dados projetados




histograma



2 erros

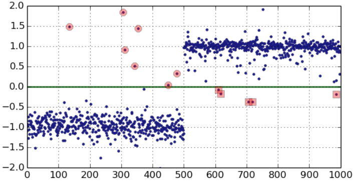
# Discriminantes Lineares

## Exemplo: Discriminantes lineares (2 classes)

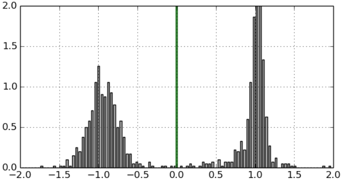
- Resultados – dados de teste

  Aplicar transformação (vector **w** - calculado com os dados de treino)

  Não re-calcular o vector **w**!

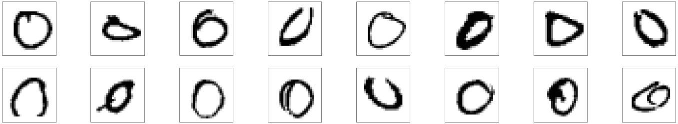



dados projetados




histograma

Total de 23 erros — 16 erros nas imagens dos $0_s$



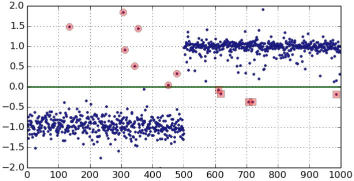
# Discriminantes Lineares

## Exemplo: Discriminantes lineares (2 classes)

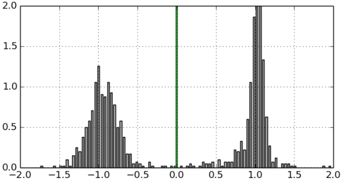
- Resultados – dados de teste

  Aplicar transformação (vector **w** - calculado com os dados de treino)

  Não re-calcular o vector **w**!




dados projetados




histograma

Total de 23 erros — 7 erros nas imagens dos $1_s$

# Discriminantes Lineares

## Exemplo: Discriminantes lineares (multi-classe)

- Objectivo: Separar imagens de dígitos manuscritos (10 classes)

1. Construir matriz X de dados de treino:

```
>>> D=pickle.load(open('MNISTsmall.p','rb'))
>>> f=D['foldTrain']
>>> y=D['trueClass'][f]
>>> X=D['X'][:,f] #matriz de 784×10000
```

acrescentar uma linha de 1s

```
>>> X=np.vstack(np.ones(10000),X) #matriz de 785×10000
```

2. Construir a matriz Y de saídas desejadas:

Inicializar matriz Y com valores −1 (menos um)

```
>>> Y=-np.ones((10,10000)) #matriz de 10×10000
```

Atribuir +1 para identificar classes:

```
>>> for i in range(10):
        Y[i,i*1000:(i+1)*1000]=1
```

3. Estimar matriz **W**

Construir $\mathbf{R_x}$ e $\mathbf{r_{x}}_y$

```
>>> Rx=np.dot(X,X.T) #matriz de 785×785
>>> rxy=np.dot(X,Y.T) #matriz de 785×10
```

Calcular **W**

```
>>> W=np.dot(np.linalg.pinv(Rx),rxy) #matriz de 785×10
```

# Discriminantes Lineares

## Exemplo: Discriminantes lineares (multi-classe)

- Objectivo: Separar imagens de dígitos manuscritos (10 classes)

  4. Classificação do **conjunto de treino**:

     Transformar os dados e determinar o maior valor dos $\hat{y}_i$

     ```
     >>> YhTrain=np.dot(W.T,X.T)
     >>> estCtrain=np.argmax(YhTrain,axis=0)
     ```

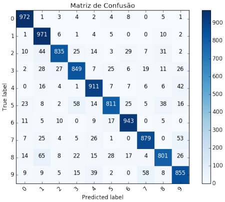
  5. Calcular matriz de confusão:

     Criar array com classes verdadeiras:

     ```
     >>> trueCtrain=np.zeros(10000)
     >>> for i in range(10):
             trueCtrain[i,i*1000:(i+1)*1000]=i
     ```

  6. Estimar matriz de confusão:

     ```
     >>> from sklearn.metrics import confusion_matrix
     >>> ConfMat=confusion_matrix(trueCtrain,estCtrain)
     ```

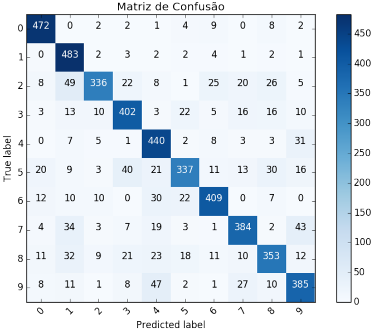



$$\text{Prob.Erro} = \frac{1173}{10000} = 11.73\%$$

# Discriminantes Lineares

## Exemplo: Discriminantes lineares (multi-classe)

- Objectivo: Separar imagens de dígitos manuscritos (10 classes)
  7. Classificação do **conjunto de teste**:
     Repetir o processo feito para os dados de treino, com os dados de teste.
  8. Calcular matriz de confusão:



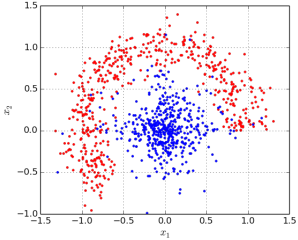

$$\text{Prob.Erro} = \frac{999}{5000} = 19.98\%$$

# Discriminantes Quadráticos

## Exemplo: Discriminantes Quadráticos (2 classes)

`QuadDiscData.p`

- Discriminantes lineares são modelos demasiado rígidos para lidarem com distribuições de pontos como as deste exemplo.

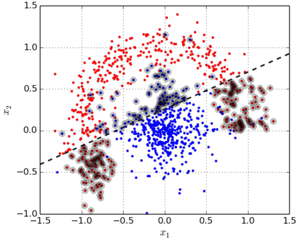


# Discriminantes Quadráticos

## Exemplo: Discriminantes Quadráticos (2 classes)

`QuadDiscData.p`

- Discriminantes lineares são modelos demasiado rígidos para lidarem com distribuições de pontos como as deste exemplo.
- Neste caso são cometidos 266 erros (num total de 1000 pontos)



# Discriminantes Quadráticos

## Exemplo: Discriminantes Quadráticos (2 classes)

`QuadDiscData.p`

- Discriminantes lineares podem ser facilmente modificados para conterem transformações não-lineares dos dados.
- Uma possível escolha de não-linearidades é incluir no modelo funções quadráticas dos dados. Neste exemplo pode-se usar o seguinte discriminante quadrático:

$$\begin{aligned}\hat{y} &= w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2 \\ \hat{y} &= \begin{bmatrix} w_0 & w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix} = \mathbf{w}^\top \mathbf{x}\end{aligned}$$

- O modelo continua a ser **linear nos parâmetros**. Isto significa que se pode estimar analiticamente o vector **w** derivando a função do erro quadrático médio, igualando a zero, e resolvendo o sistema de equações resultante.
- Solução (é a mesma expressão): $\mathbf{w} = \mathbf{R}_{\mathbf{x}}^{-1} \mathbf{r}_{\mathbf{x}y}$

# Discriminantes Quadráticos

## Exemplo: Discriminantes Quadráticos (2 classes)

`QuadDiscData.p`

Instruções em Python:

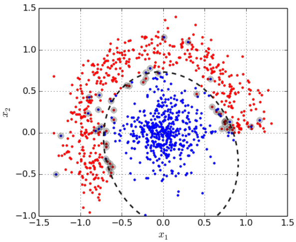
1. Construir matriz `X` (dados 2D, previamente guardados na matrix `data` $2\times1000$):
```
>>> x1,x2=(data[0,:],data[1,:])
>>> X=np.vstack((np.ones(1000),x1,x2,x1**2,x2**2,x1*x2))
```
2. Matriz `Y`:
```
>>> Y=np.hstack((-np.ones(500),np.ones(500)))
```
3. Estimar vector `w`:
```
>>> Rx=np.dot(X,X.T)
>>> rxy=np.dot(X,Y.T)
>>> Rxi=np.linalg.pinv(Rx)
>>> w=np.dot(Rxi,rxy)
```
4. Classificar:
```
>>> Yh=np.dot(w.T,X)
>>> estC=Yh>=0
```

# Discriminantes Quadráticos

## Exemplo: Discriminantes Quadráticos (2 classes)

`QuadDiscData.p`

- Discriminantes quadráticos já conseguem lidar com situações em que as fronteiras de decisão são curvas.
- Neste caso são cometidos 45 erros (num total de 1000 pontos).



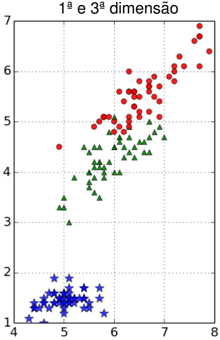

NOTA: Este exemplo serve para evidenciar as limitações dos discriminantes lineares, e para ilustrar como algumas destas limitações podem ser ultrapassadas usando discriminantes quadráticos. No entanto, os modelos foram estimados e testados com o mesmo conjunto de pontos: para ter uma estimativa mais acertada do erro será necessário usar outra metodologia de teste (ex: validação cruzada).

# Discriminantes Quadráticos

## Exemplo: Discriminantes Quadráticos (multi-classes)

Base de dados `Iris` do `scikit-learn`

- Discriminantes lineares são modelos demasiado rígidos para lidarem com distribuições de pontos como as deste exemplo.
- Os pontos da segunda classe (a verde) encontram-se entre os pontos das outras duas. Isto faz com que o discriminante linear relativo a esta classe seja mal estimado.

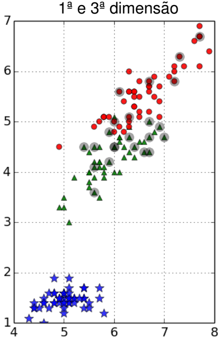


# Discriminantes Quadráticos

## Exemplo: Discriminantes Quadráticos (multi-classes)

Base de dados `Iris` do `scikit-learn`

- Discriminantes lineares são modelos demasiado rígidos para lidarem com distribuições de pontos como as deste exemplo.
- Os pontos da segunda classe (a verde) encontram-se entre os pontos das outras duas. Isto faz com que o discriminante linear relativo a esta classe seja mal estimado.
- Neste caso, são cometidos 23 erros (num total de 150 pontos), dos quais 16 são dados da classe 2 classificados na 3.



# Discriminantes Quadráticos

## Exemplo: Discriminantes Quadráticos (multi-classes)

Base de dados `Iris` do `scikit-learn`

- Para dados com 4 (quatro) dimensões, um discriminante quadrático tem a seguinte formulação:

$$\begin{aligned}\hat{y} = \quad & w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_1^2 + w_6 x_2^2 + w_7 x_3^2 + w_8 x_4^2 \\ & + w_9 x_1 x_2 + w_{10} x_1 x_3 + w_{11} x_1 x_4 + w_{12} x_2 x_3 + w_{13} x_2 x_4 + w_{14} x_3 x_4\end{aligned}$$

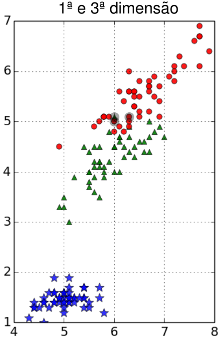
  É necessário estimar 15 parâmetros.
- Neste exemplo existem 3 classes, por isso é necessário um discriminante quadrático por classe. Isto resulta numa matriz $\mathbf{W}^\top$ de $3 \times 15$
- Solução (é a mesma expressão): $\mathbf{W} = \mathbf{R}_\mathbf{x}^{-1} \mathbf{r}_{\mathbf{x}y} = \left(\mathbf{X}\mathbf{X}^\top\right)^{-1} \mathbf{X}\mathbf{Y}^\top$
- Matriz de dados $\mathbf{X}$ de $15 \times 150$. Cada coluna desta matriz é um vector $\mathbf{x} = [1, x_1, x_2, x_3, x_4, x_1^2, x_2^2, x_3^2, x_4^2, x_1 x_2, x_1 x_3, x_1 x_4, x_2 x_3, x_2 x_4, x_3 x_4]^\top$
- Matriz de classes $\mathbf{Y}$ de $3 \times 150$ (com $\pm 1_s$ ).

# Discriminantes Quadráticos

## Exemplo: Discriminantes Quadráticos (multi-classes)

Base de dados `Iris` do `scikit-learn`

- Discriminantes quadráticos já conseguem lidar com situações como a deste exemplo.
- Neste caso, já só são cometidos 3 erros (num total de 150 pontos).
- O número de parâmetros a estimar aumenta exponencialmente com o número de dimensões dos dados, o que inviabiliza a utilização destes discriminantes em dados com elevada dimensão.




NOTA: Os modelos foram estimados e testados com o mesmo conjunto de pontos: para ter uma estimativa mais acertada do erro será necessário usar outra metodologia de teste (ex: validação cruzada).