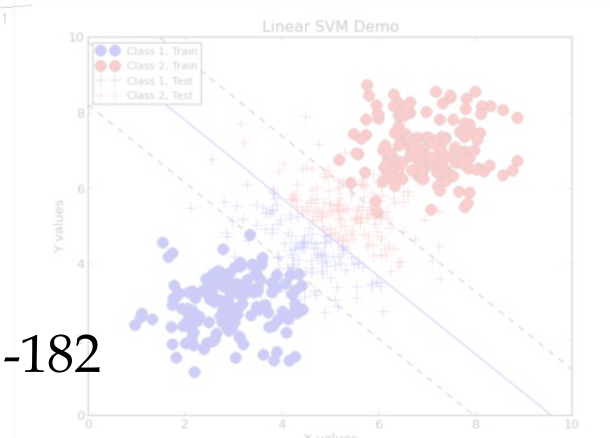
Universidade Federal do Paraná (UFPR)
Bacharelado em Informática Biomédica

Classificadores Lineares



David Menotti
www.inf.ufpr.br/menotti/ci171-182

Hoje

- Funções Discriminantes Lineares
  - Perceptron
  - Support Vector Machines (SVM)

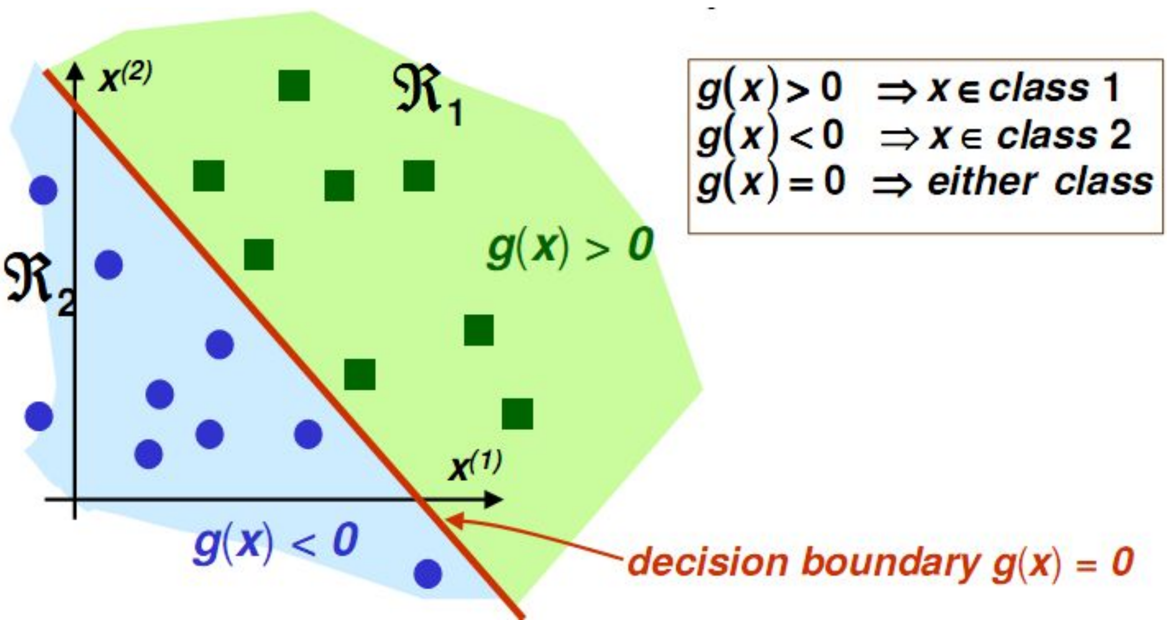
Funções Discriminantes Lineares

Funções Discriminantes Lineares

- Em geral, uma função discriminante linear pode ser escrita na forma

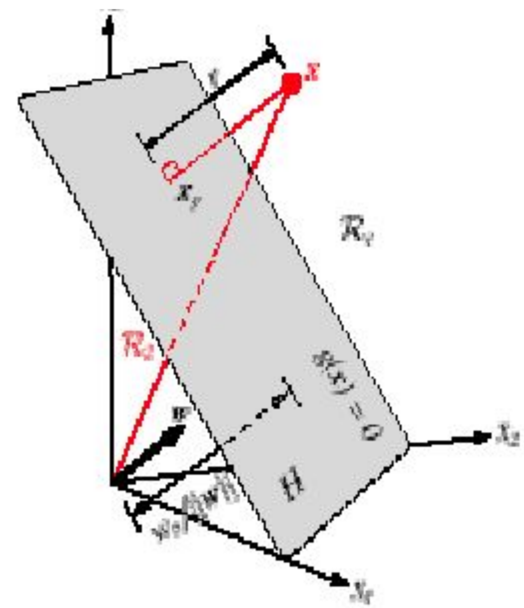
$$g(x) = \omega^T x + \omega_0$$

- ω é conhecido como o vetor dos pesos e ω_0 representa o *bias*



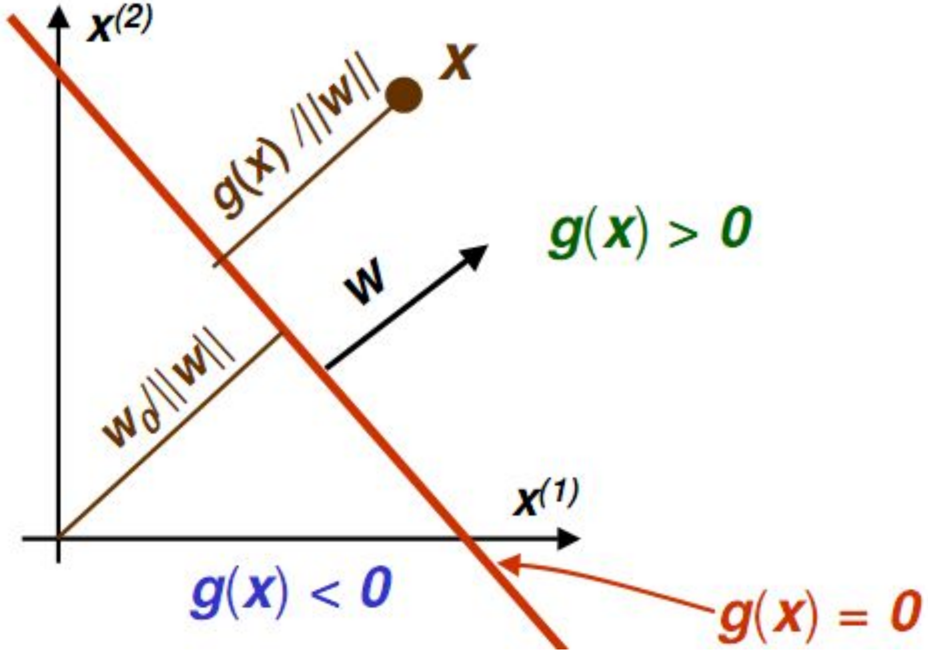
Funções Discriminante Lineares

- $g(x) = \omega^T x + \omega_0$ é um hiperplano
  - Um hiperplano é
    - Um ponto em 1D
    - Uma reta em 2D
    - Um plano em 3D



Funções Discriminante Lineares

- Para duas dimensões,
  - ω determina a orientação do hiperplano
    ω_0 representa o deslocamento (origem)



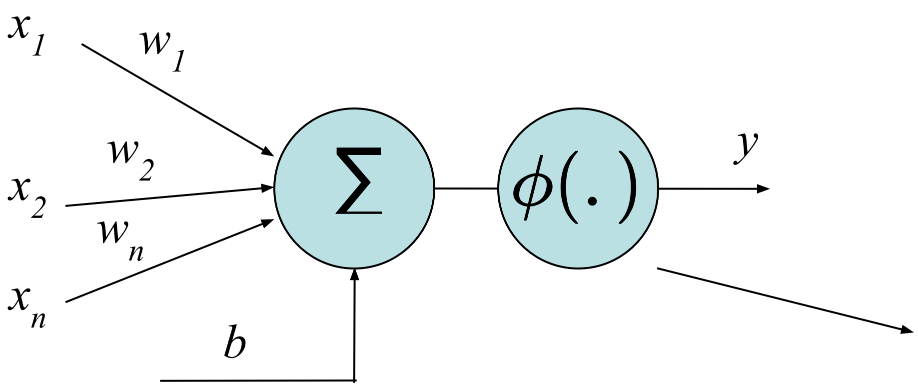
Funções Discriminante Lineares

Perceptron

Perceptron

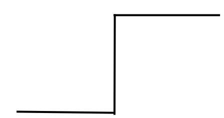
- Um classificador linear bastante simples, mas bastante importante no desenvolvimento das redes neuronais é o Perceptron.
  - O perceptron é considerado como sendo a primeira e mais primitiva estrutura de rede neuronal artificial.
  - Concebido por McCulloch and Pits na década de 50.
- Diferentemente do ***LDA***, o perceptron não transforma os dados para fazer classificação.
  - Tenta encontrar a melhor fronteira linear que separa os dados.

Perceptron



A função de ativação normalmente utilizada no perceptron é a **hardlim** (*threshold*)

$$\phi(z) = \begin{cases} 1 & \text{se } z \geq 0, \\ 0 & \text{se } z < 0 \end{cases}$$



$$y = \phi\left(b + \sum_i \omega_i x_i\right)$$

A função de ativação é responsável por determinar a forma e a intensidade de alteração dos valores transmitido de um neurônio a outro.

Perceptron
Algoritmo de Aprendizagem

1. Iniciar os pesos e bias com valores pequenos, Geralmente no intervalo [0,3 ; 0,8]
2. Aplicar um padrão de entrada com seu respectivo valor desejada de saída (t_j) e verificar a saída (y) da rede
3. Calcular o erro da saída $$e = t_j - y$$
4. Se e = 0, volta ao passo 2
5. Se e <> 0,
   - Atualizar os pesos $$\omega_i = \omega_i^{old} + e \times x_i$$
   - Atualizar o *bias* $$b = b^{old} + e$$
6. Voltar ao passo 2

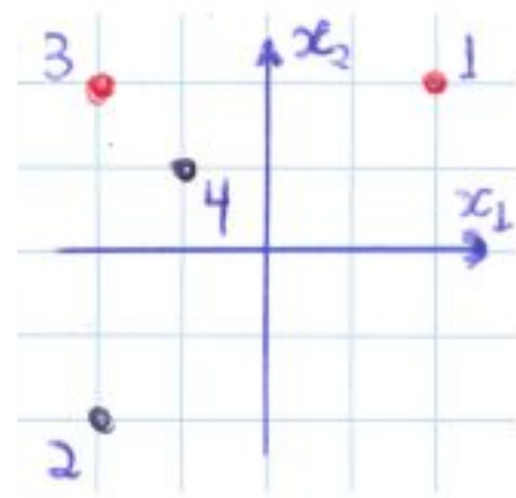
Critério de parada:

Todos os padrões classificados corretamente.

Perceptron: Exemplo

- Considere o seguinte conjunto de aprendizagem.

X		*t*
2	2	0
-2	-2	1
-2	2	0
-1	1	1

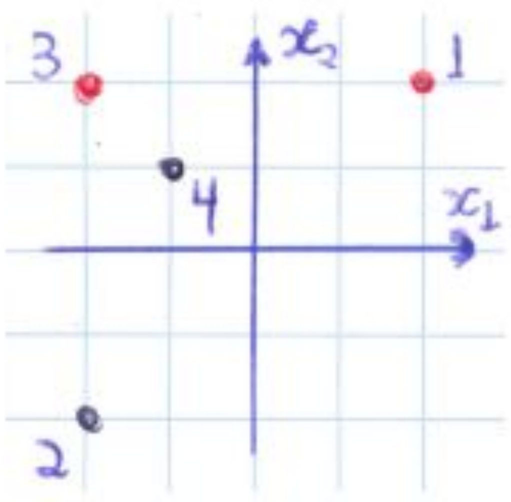


Nesse tipo de algoritmo é importante que os dados sejam apresentados ao algoritmo de treinamento de maneira intercalada (*shuffle*)

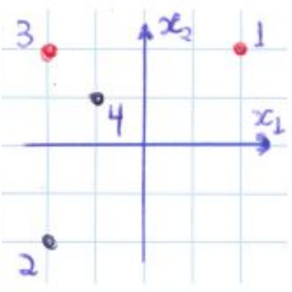
Perceptron: Exemplo

Exemplo

- **Nesse** exemplo, vamos inicializar:
  - $\omega = [0, 0]$ e $b = 0$
- E a Função Discriminante?
  - Fica indefinida!
    - $g(x) = \omega^T x + b = 0$
    - $0x_1 + 0x_2 = 0$
  - Primeira amostra definirá um hiperlano



Perceptron: Exemplo



Exemplo

- Apresentando o primeiro padrão (X_1) a rede.

$$y = hardlim([0,0][2,2]^T + 0) = hardlim(0) = 1$$

- Calcula-se o erro $$e = t_i - y = 0 - 1 = -1$$
- Como o erro é diferente de 0, atualiza-se os pesos e o bias

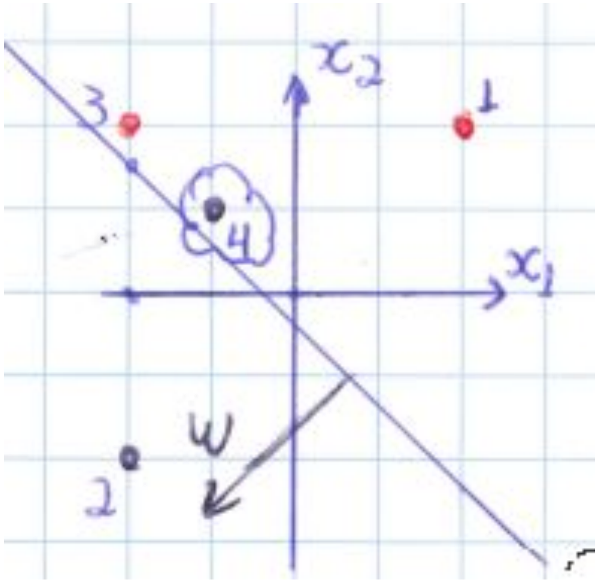
$$\omega = \omega^{old} + e \times x_i = [0,0] + (-1[2,2]) = [-2,-2]$$

$$b = b^{old} + e = 0 + (-1) = -1$$

Perceptron

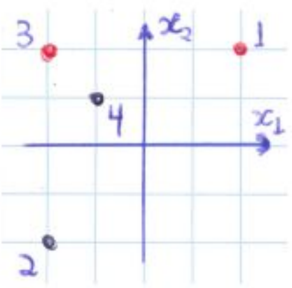
Exemplo

$$-2x_1 - 2x_2 - 1 \geq 0$$



$$\omega = [-2, -2], b = -1$$

Perceptron



Exemplo

- Apresentando o segundo padrão (X_2) a rede:

$$y = hardlim\left([-2,-2][-2,-2]^T + (-1)\right) = hardlim(7) = 1$$

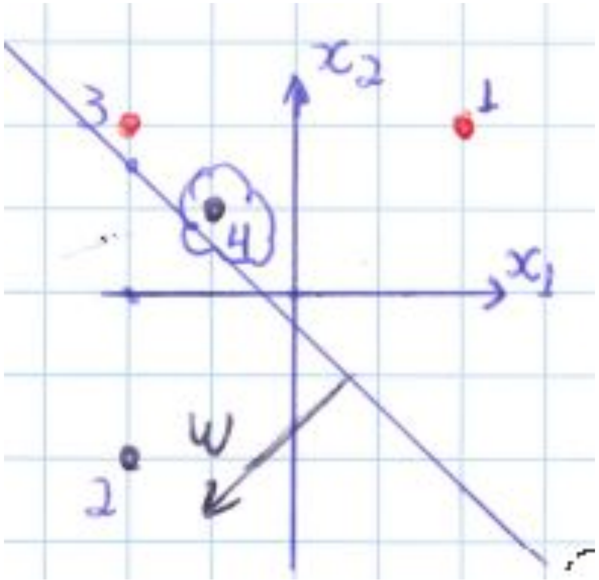
- Calcula-se o erro $e = t_i - y = 1 - 1 = 0$

- Como o erro é 0, os pesos e o bias não precisam ser atualizados.

Perceptron

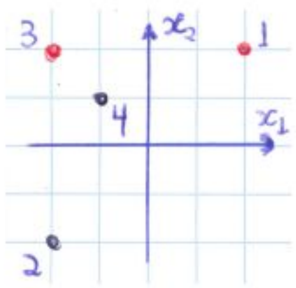
Exemplo

$$-2x_1 - 2x_2 - 1 \geq 0$$



$$\omega = [-2, -2], b = -1$$

Perceptron



Exemplo

- Apresentando o terceiro padrão (X_3) a rede:

$$y = hardlim\left([-2,-2][-2,+2]^T + (-1)\right) = hardlim(-1) = 0$$

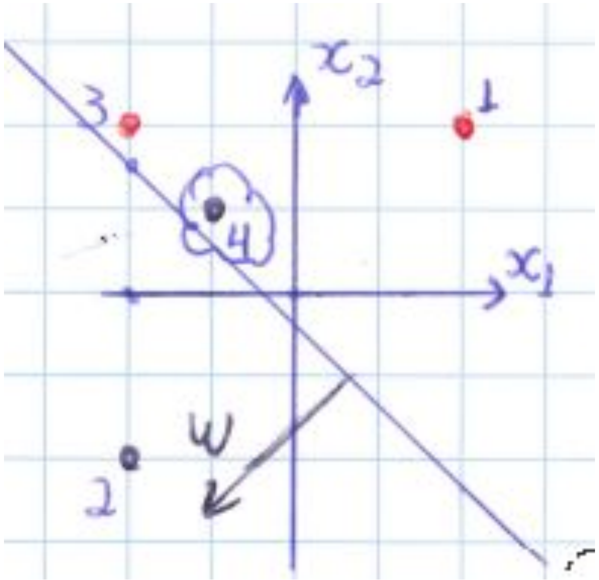
- Calcula-se o erro $e = t_i - y = 0 - 0 = 0$

- Como o erro é 0, os pesos e o *bias* não precisam ser atualizados.

Perceptron

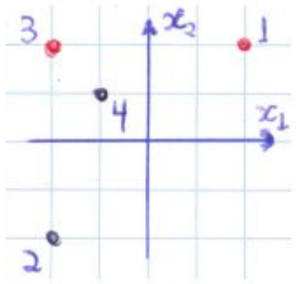
Exemplo

$$-2x_1 - 2x_2 - 1 \geq 0$$



$$\omega = [-2, -2], b = -1$$

Perceptron



Exemplo

- Apresentando o quarto padrão (X_4) a rede:

$$y = hardlim\left([-2,-2][-1,+1]^T + (-1)\right) = hardlim(-1) = 0$$

- Calcula-se o erro $e = t_i - y = 1 - 0 = 1$

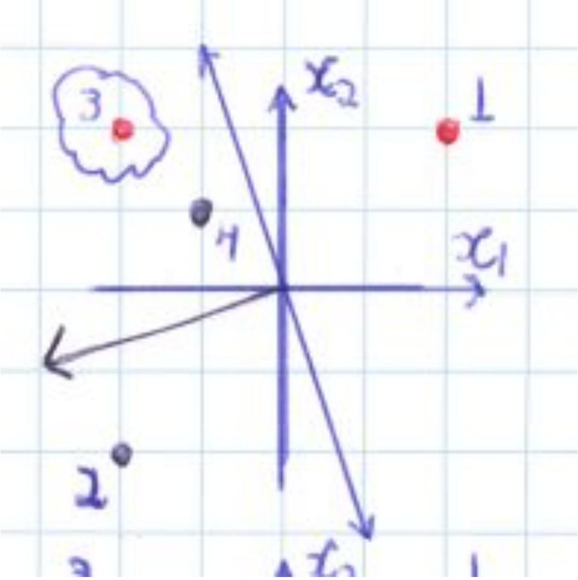
- Como o erro é diferente de 0, atualiza-se os pesos e o bias

$$\omega = \omega^{old} + e \times x_i = [-2,-2] + (+1[-1,1]) = [-3,-1]$$
$$b = b^{old} + e = -1 + 1 = 0$$

Perceptron

Exemplo

$$-3x_1 - 1x_2 - 1 \geq 0$$



$$\omega = [-3, -1], b = 0$$

Perceptron

Exemplo

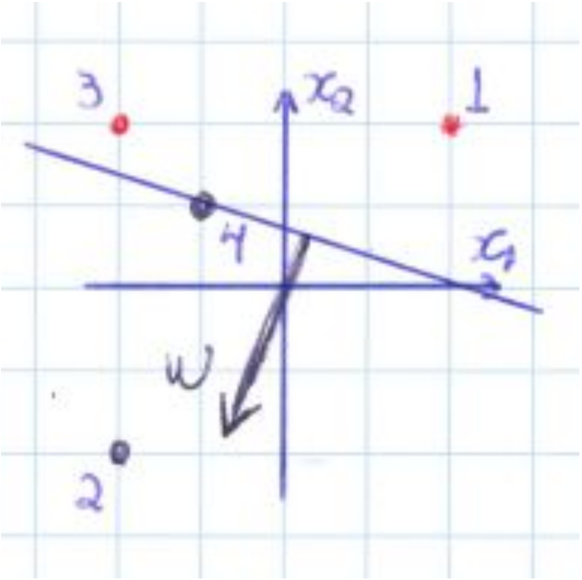
- O processo evolui por várias iterações

it: 1, i: 0, X=[2.00, 2.00], t= 0.00, y = 1.00, e = -1.00
W=[-2.00, -2.00], B= -1.00
it: 2, i: 1, X=[-2.00, -2.00], t= 1.00, y = 1.00, e = 0.00
it: 3, i: 2, X=[-2.00, 2.00], t= 0.00, y = 0.00, e = 0.00
it: 4, i: 3, X=[-1.00, 1.00], t= 1.00, y = 0.00, e = 1.00
W=[-3.00, -1.00], B= 0.00
it: 5, i: 0, X=[2.00, 2.00], t= 0.00, y = 0.00, e = 0.00
it: 6, i: 1, X=[-2.00, -2.00], t= 1.00, y = 1.00, e = 0.00
it: 7, i: 2, X=[-2.00, 2.00], t= 0.00, y = 1.00, e = -1.00
W=[-1.00, -3.00], B= -1.00
it: 8, i: 3, X=[-1.00, 1.00], t= 1.00, y = 0.00, e = 1.00
W=[-2.00, -2.00], B= 0.00
it: 9, i: 0, X=[2.00, 2.00], t= 0.00, y = 0.00, e = 0.00
it: 10, i: 1, X=[-2.00, -2.00], t= 1.00, y = 1.00, e = 0.00
it: 11, i: 2, X=[-2.00, 2.00], t= 0.00, y = 1.00, e = -1.00
W=[0.00, -4.00], B= -1.00
it: 12, i: 3, X=[-1.00, 1.00], t= 1.00, y = 0.00, e = 1.00
W=[-1.00, -3.00], B= 0.00
it: 13, i: 0, X=[2.00, 2.00], t= 0.00, y = 0.00, e = 0.00
it: 14, i: 1, X=[-2.00, -2.00], t= 1.00, y = 1.00, e = 0.00
it: 15, i: 2, X=[-2.00, 2.00], t= 0.00, y = 0.00, e = 0.00
it: 16, i: 3, X=[-1.00, 1.00], t= 1.00, y = 0.00, e = 1.00
W=[-2.00, -2.00], B= 1.00
it: 17, i: 0, X=[2.00, 2.00], t= 0.00, y = 0.00, e = 0.00
it: 18, i: 1, X=[-2.00, -2.00], t= 1.00, y = 1.00, e = 0.00
it: 19, i: 2, X=[-2.00, 2.00], t= 0.00, y = 1.00, e = -1.00
W=[0.00, -4.00], B= 0.00
it: 20, i: 3, X=[-1.00, 1.00], t= 1.00, y = 0.00, e = 1.00
W=[-1.00, -3.00], B= 1.00
it: 21, i: 0, X=[2.00, 2.00], t= 0.00, y = 0.00, e = 0.00
it: 22, i: 1, X=[-2.00, -2.00], t= 1.00, y = 1.00, e = 0.00
it: 23, i: 2, X=[-2.00, 2.00], t= 0.00, y = 0.00, e = 0.00
it: 24, i: 3, X=[-1.00, 1.00], t= 1.00, y = 0.00, e = 1.00
W=[-2.00, -2.00], B= 2.00
it: 25, i: 0, X=[2.00, 2.00], t= 0.00, y = 0.00, e = 0.00
it: 26, i: 1, X=[-2.00, -2.00], t= 1.00, y = 1.00, e = 0.00
it: 27, i: 2, X=[-2.00, 2.00], t= 0.00, y = 1.00, e = -1.00
W=[0.00, -4.00], B= 1.00
it: 28, i: 3, X=[-1.00, 1.00], t= 1.00, y = 0.00, e = 1.00
W=[-1.00, -3.00], B= 2.00
it: 29, i: 0, X=[2.00, 2.00], t= 0.00, y = 0.00, e = 0.00
it: 30, i: 1, X=[-2.00, -2.00], t= 1.00, y = 1.00, e = 0.00
it: 31, i: 2, X=[-2.00, 2.00], t= 0.00, y = 0.00, e = 0.00
it: 32, i: 3, X=[-1.00, 1.00], t= 1.00, y = 1.00, e = 0.00

Perceptron

- O processo acaba quando todos os padrões forem classificados corretamente.

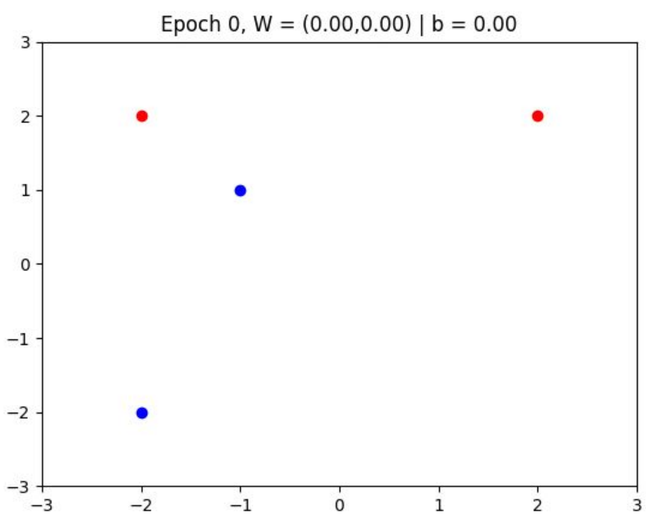
$$-1x_1 - 3x_2 + 2 \geq 0$$

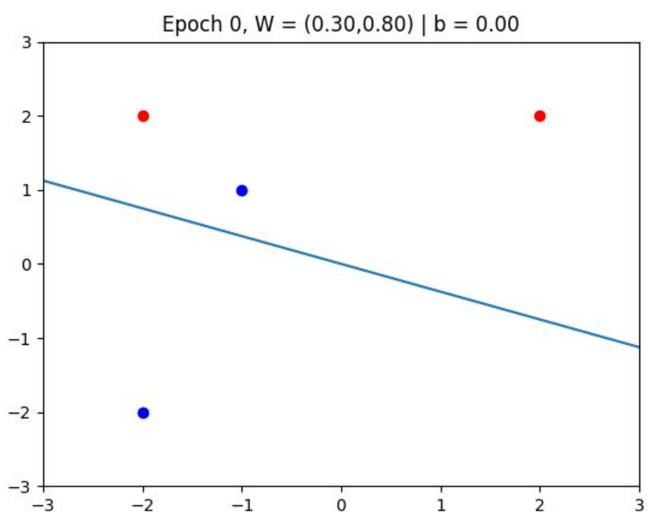


$$\omega = [-1, -3], b = 2$$

Perceptron

Exemplo



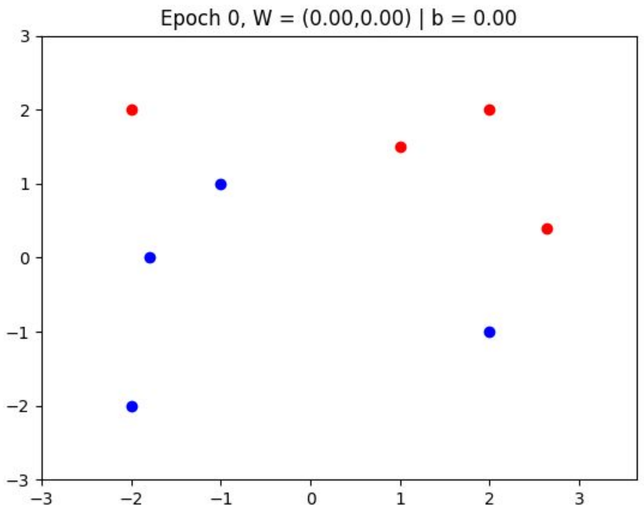


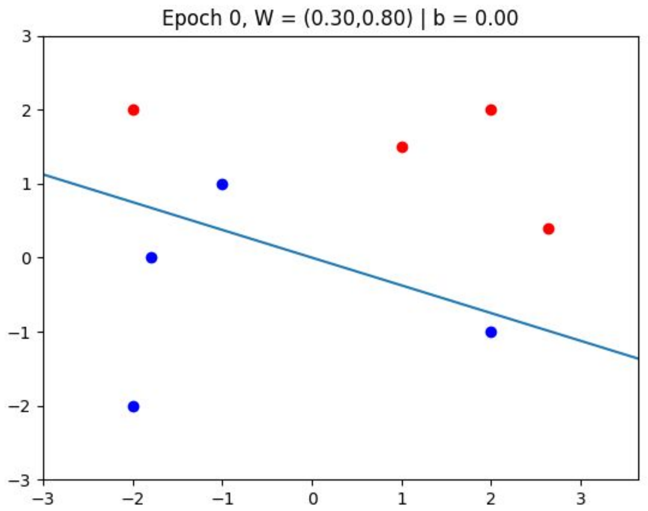
$$0 : \omega = [0.0, 0.0] \; b = 0.0$$

$$0 : \omega = [0.3, 0.8] \; b = 0.0$$

Perceptron

Exemplo - 8 pontos





$$0 : \omega = [0.0, 0.0]\ b = 0.0$$

$$0 : \omega = [0.3, 0.8]\ b = 0.0$$

Determinando a fronteira

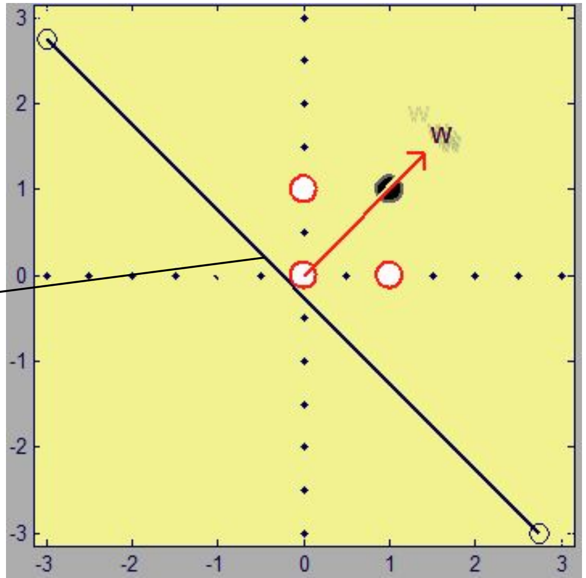
- No caso bi-dimensional, a fronteira de decisão pode ser facilmente encontrada usando a seguinte equação

$$x_2 = \frac{-\omega_1 \times x_1 - b}{\omega_2} \qquad \omega_1 \times x_1 + \omega_2 \times x_2 + b = 0$$

Considere o seguinte exemplo, $\omega = [1,41; 1,41], b = 0,354$

Escolha duas coordenadas x_1, para então encontrar os x_2 correspondentes

X = [-3,3]



Efeito do bias
Diferente de zero.

Para x_1 = -3, x_2 = 2,74
Para x_1 = +3, x_2 = -3,25

Perceptron

Código fonte da simulação

```c
#include <stdio.h>

#define TAM 4

double hardlim(double* X,double* W, double B)
{
    double sum = B; int i;
    for(i=0;i<2;i++)
        sum+= W[i] * X[i];
    return ( sum >= 0. ) ? 1. : 0.;
}
```

```c
int main(void){
    int c,n,i,it;
    double y,e;
    double X[][2] = {{2.,2.},{-2.,-2.},{-2.,2.},{-1.,1.}};
    double T[] = {0.,1.,0.,1.};

    double W[] = {0.,0.}, B = 0.;

    it=0;n = 0;c=0;
    while (c<4) {
        it++;

        y = hardlim(X[n],W,B);
        e = T[n] - y;

        printf("it:%3d, i:%2d, X=[%6.2lf,%6.2lf], "
          , it,n,X[n][0],X[n][1]);
        printf("t=%6.2lf, y = %6.2lf, e = %6.2lf\n"
          ,T[n],y,e);

        if ( e != 0. ) {
            for(i=0;i<2;i++) {
                W[i] = W[i] + e*X[n][i];
            }
            B = B + e;
            printf("W=[%6.2lf,%6.2lf],",W[0],W[1]);
            printf("B=%6.2lf\n"        ,B);
            c=0;
        }
        else
            c++;
        n = (n==TAM-1) ? 0 : n+1;
    }
}
```

Perceptron

Código fonte da visualização

```python
#!/usr/bin/python # -*- encoding: iso-8859-1 -*-
# Aprendizagem de Máquina / Rayson Bartoski Laroca dos Santos
import sys,math,time,pylab
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_svmlight_file, make_classification

i = 0
def predict(row, weights):
    activation = weights[0]
    for i in range(len(row)):
        activation += weights[i + 1] * row[i]
    # hardlim
    return 1.0 if activation >= 0.0 else 0.0

def update_weights(weights, error, X):
    global i

    w1 = weights[1] + error * X[i][0]
    w2 = weights[2] + error * X[i][1]
    bias = weights[0] + error

    return bias,w1,w2

def plot_line(X,y,weights,epoch,time):
    for x,label in zip(X,y):
        if label == 0:
            plt.scatter(x[0],x[1], color='red')
        else:
            plt.scatter(x[0],x[1], color='blue')

    # equação da reta - x * w1 + y * w2 + bias = 0
    xx = []
    yy = []
    x_min = min(X.transpose()[0])
    x_max = max(X.transpose()[0])
    y_min = min(X.transpose()[1])
    y_max = max(X.transpose()[1])

    # point 1
    x = x_max + 1
    if weights[2] == 0:
        y = 0
    else:
        y = (x * weights[1] + weights[0]) / - weights[2]
    xx.append(x)
    yy.append(y)

    # point 2
    x = x_min - 1
    if weights[2] == 0:
        y = 0
    else:
        y = (x * weights[1] + weights[0]) / - weights[2]
    xx.append(x)
    yy.append(y)

    # plot perceptron line
    fig = pylab.gcf()
    fig.canvas.set_window_title('Epoch ' + str(epoch))

    if weights[1] <> 0 and weights[2] <> 0:
        plt.plot(xx,yy)
    plt.title('Epoch {0:1d}, W = ({1:.2f},{2:.2f}) | b = {3:.2f}'
         .format(epoch,weights[1],weights[2],weights[0]))
    plt.xlim(x_min-1,x_max+1)
    plt.ylim(y_min-1,y_max+1)
    fig.savefig('./perceptron'+str(epoch))

    # updates every x seconds
    plt.pause(time)
    plt.clf()

def main(data):
    global i

    t = 0.50 # tempo entre um plot e outro.
    t_final = 2 # tempo que o plot fica no final

    if data == None:
        print 'No dataset given | Usage: perceptron.py <data>'
        # print 'Loading default data...'
        # time.sleep(2)
        # X = np.array([[2,2],[-2,-2],[-2,2],[-1,1],[1,1.5],[2,-1],[2.64,
        # y = np.float64([0,1,0,1,0,1,0,1])

        print 'Loading random data...'
        time.sleep(2)
        X, y = make_classification(n_samples=30, n_features=2,
                                   n_redundant=0, n_clusters_per_class=1)
    else:
        print "Loading data..."
        X, y = load_svmlight_file(data)
        X = X.toarray()

    weights = [0, 0.3, 0.8]
    #weights = [0, 0, 0]

    print 'W = ({0:.2f},{1:.2f}) | b = {2:.2f}'.format(weights[1],weights[

    epoch = 0
    correct = 0

    plot_line(X,y,weights,epoch, t)
    while correct < len(X) and epoch < 25:
        if i == len(X):
            i = 0
        prediction = predict(X[i], weights)
        gt = y[i]
        print("  Pattern %d -> gt=%d, Predicted=%d" % (i, gt, prediction))
        if gt == prediction:
            correct += 1
        else:
            error = gt - prediction

            weights = update_weights(weights, error, X)

            print '\nUpdate Weights:'
            print '  Weights = ({0:.2f},{1:.2f}) | bias = {2:.2f}'.format(weight

            # i = 0
            correct = 0

        if i == 0:
            epoch += 1
            plot_line(X,y,weights,epoch, t)

        i += 1

    print '\nResults:'
    print '  Epochs:', epoch
    print '  Weights = ({0:.2f},{1:.2f}) | bias = {2:.2f}'.format(weights[1],wei
    plot_line(X,y,weights,epoch,t_final)

if __name__ == "__main__":
    time1 = time.time()

    if len(sys.argv) == 1:
        main(None)
    elif len(sys.argv) != 2:
        sys.exit("Usage: python perceptron.py data.dat")
    else:
        main(sys.argv[1])

    time2 = time.time()
    seconds = (time2-time1)
    m, s = divmod(seconds, 60)
    h, m = divmod(m, 60)
```

Perceptron

Usando um Perceptron em python

Load the required elements

```python
import numpy as np
import matplotlib.pyplot as plt
#import sklearn.linear_model.perceptron as p
from sklearn.linear_model import perceptron

# Needed to show the plots inline
%matplotlib inline
```

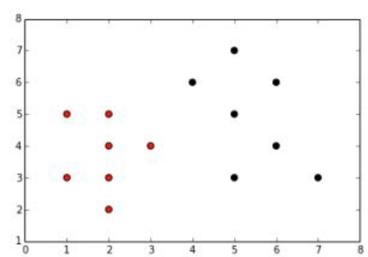
Set the data

```python
# Data
d = np.array([
[2, 1, 2, 5, 7, 2, 3, 6, 1, 2, 5, 4, 6, 5],
[2, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 7]
])

# Labels
t = np.array([0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1])
```

Plot the Data (see what it looks like)

```python
colormap = np.array(['r', 'k'])
plt.scatter(d[0], d[1], c=colormap[t], s=40)
```



Example Data

Perceptron

Usando um Perceptron em python

- Treinando e classificando

```python
# rotate the data 180 degrees
d90 = np.rot90(d)
d90 = np.rot90(d90)
d90 = np.rot90(d90)

# Create the model
net = perceptron.Perceptron(n_iter=100, verbose=0, random_state=None, fit_intercept=True, eta0=0.002)
net.fit(d90,t)

# Print the results
print "Prediction " + str(net.predict(d90))
print "Actual     " + str(t)
print "Accuracy   " + str(net.score(d90, t)*100) + "%"
```

Levando os dados para formato linha

As we can see, the model has *solved* the problem.

```
Prediction [0 0 0 1 1 0 0 1 0 0 1 1 1 1]
Actual     [0 0 0 1 1 0 0 1 0 0 1 1 1 1]
Accuracy   100.0%
```

Perceptron

Usando um Perceptron em python

- Visualizando a função criada

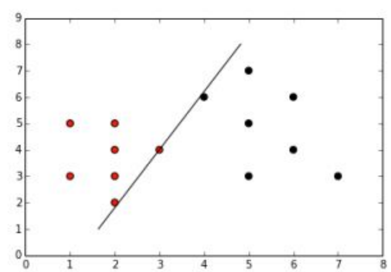
Plot the Decision Boundary

```python
# Plot the original data
plt.scatter(d[0], d[1], c=colormap[t], s=40)

# Output the values
print "Coefficient 0 " + str(net.coef_[0,0])
print "Coefficient 1 " + str(net.coef_[0,1])
print "Bias " + str(net.intercept_)

# Calc the hyperplane (decision boundary)
ymin, ymax = plt.ylim()
w = net.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(ymin, ymax)
yy = a * xx - (net.intercept_[0]) / w[1]

# Plot the line
plt.plot(yy,xx, 'k-')
```



Decision Boundary – Values on either side of the line are in different groups

Funções Discriminante Lineares

Support Vector Machines (SVM)

SVM

- Proposto por Vladimir Vapnik / Histórico
  - Introdução (1963)
  - Kernel Trick (1992)
  - Soft Margin (1995)

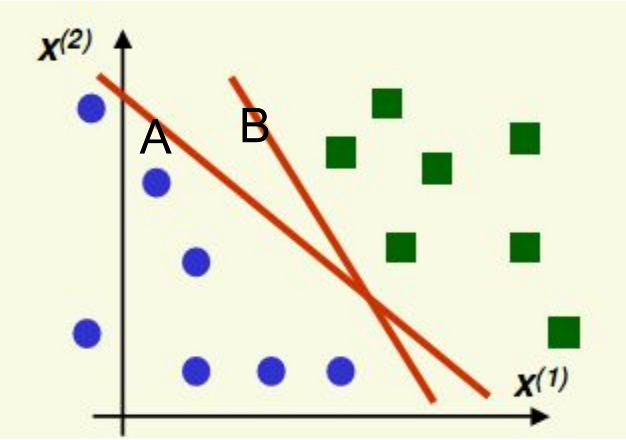
- Um dos mais importantes acontecimentos na área de reconhecimento de padrões nos últimos 25 anos.

- Tem sido largamente utilizado com sucesso para resolver diferentes problemas.

- Vapnik & Lerner (163). Pattern Recognition using Generalized Portrait Method, ARC, 24.
- Boser, Guyon & Vapnik (1992). "A training algorithm for optimal margin classifiers" – *COLT '92*.
- Cortes, C. & Vapnik (1995) V. N. Support-vector Networks. Machine Learning 20 (3), 1995

SVM - Introdução

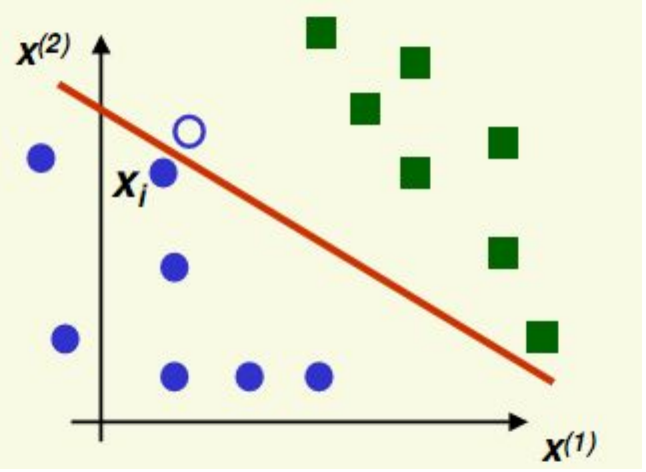
- Como vimos anteriormente, o perceptron é capaz de construir uma fronteira se os dados forem linearmente separáveis.



Mas qual a fronteira que deve ser escolhida??

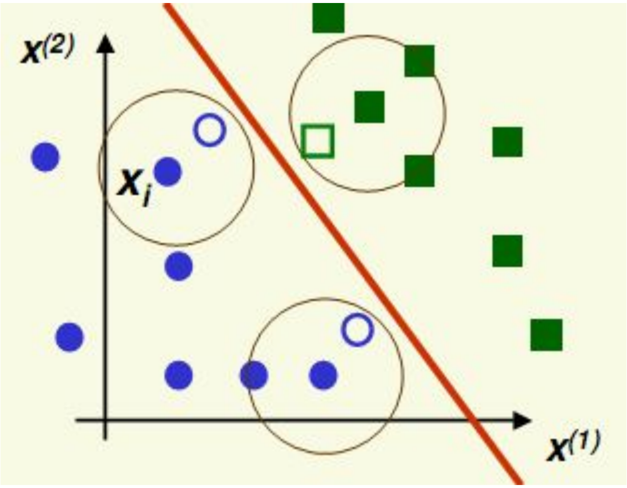
SVM - Introdução

- Suponha que a fronteira escolhida é a A.
- Como ela está bem próxima da classe azul, seu **poder de generalização é baixo**
  - Note que um novo elemento (dados não usados no treinamento), bem próximo de um azul será classificado erroneamente.



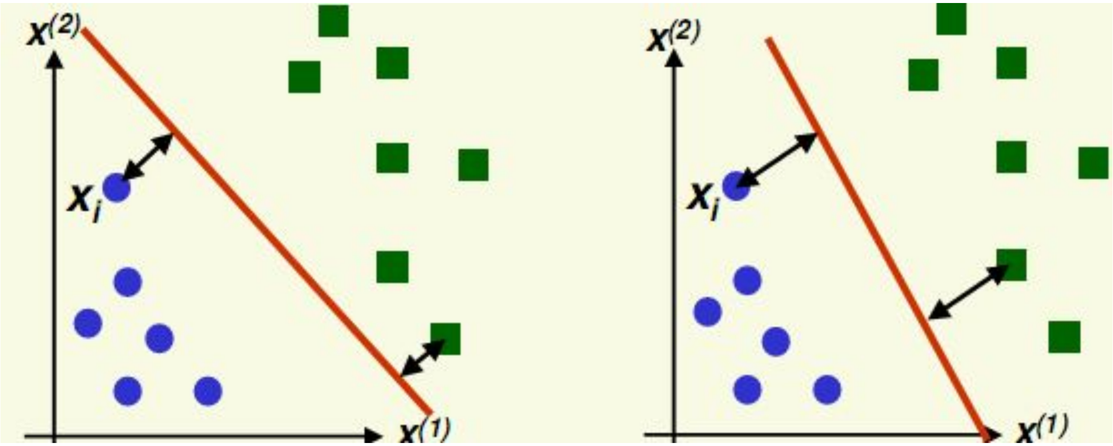
SVM - Introdução

- Escolhendo a fronteira B, podemos notar que o poder de generalização é bem melhor.
- Novos dados são corretamente classificados, pois temos uma fronteira mais distante dos dados de treinamento.



Maximização da Margem

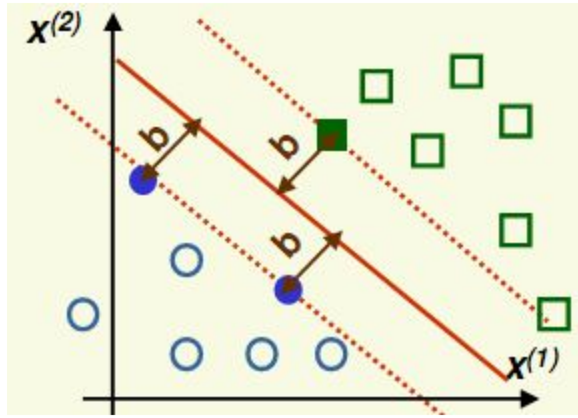
- O conceito por trás do SVM é a **maximização da margem**, ou seja, maximizar a distância da margem dos dados de treinamento



Distância Pequena   Distância Grande

Hiperplano ótimo: Distância da margem para o exemplo da classe positiva é igual a distância da margem para o exemplo da classe negativa.

Vetores de Suporte



- São os exemplos da base de treinamento mais próximos do hiperplano.
  – O hiperplano é definido unicamente pelos vetores de suporte, os quais são encontrados durante o treinamento.
  – Minimização de uma função quadrática / custo cúbico
    - Alto custo computacional.

SVM: Decisão

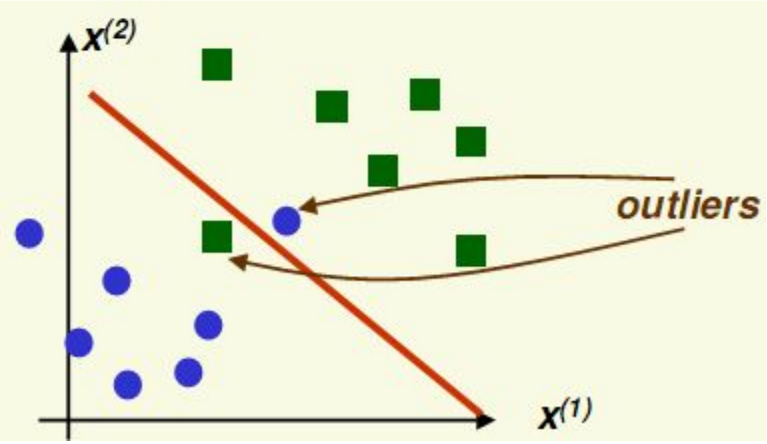
$$f(x) = \sum_i \alpha_i y_i K(x, x_i) + b$$

- A função de decisão pode ser descrita pela fórmula acima, na qual,
  - *K* é a função de **kernel**,
  - ***α*** e ***b*** são os parâmetros encontrados durante o treinamento,
  - x_i e y_i são:
    - os vetores de características e
    - o *label* da classe,

  respectivamente.

Soft Margin

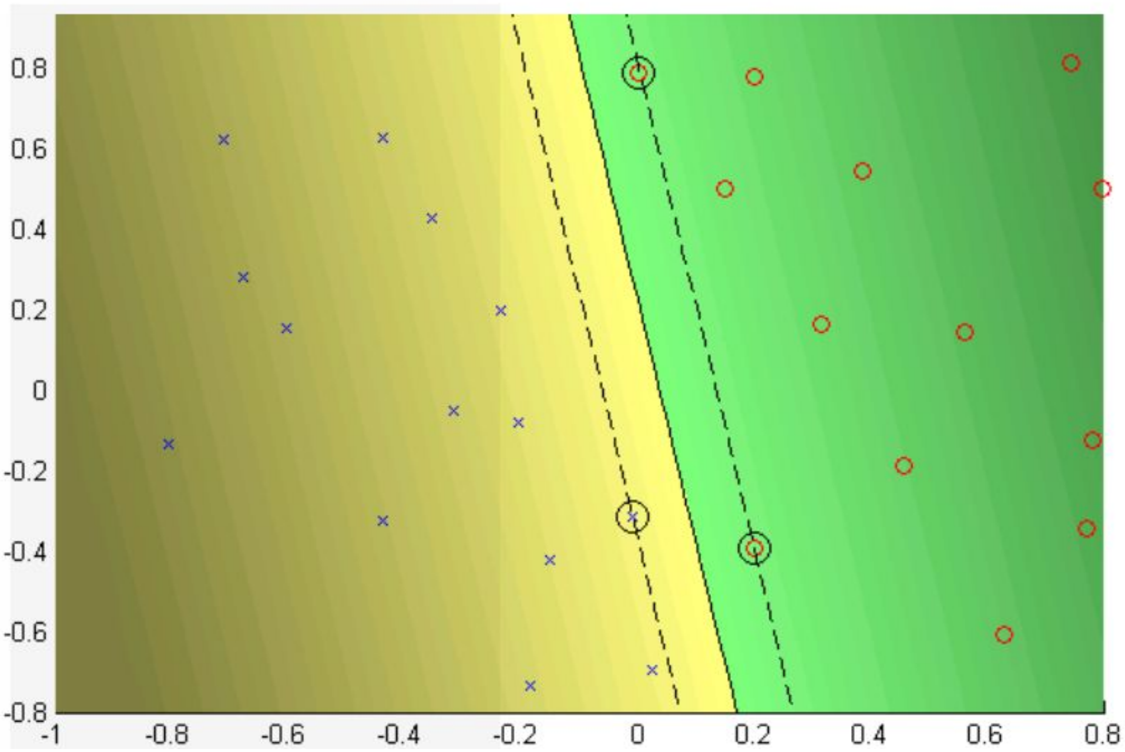
- Mesmo para dados que não podem ser separados linearmente, o SVM ainda pode ser apropriado.
- Isso é possível através do uso das "variáveis de folga" (parâmetro C).



Para um bom desempenho, os dados devem ser "quase" linearmente separáveis

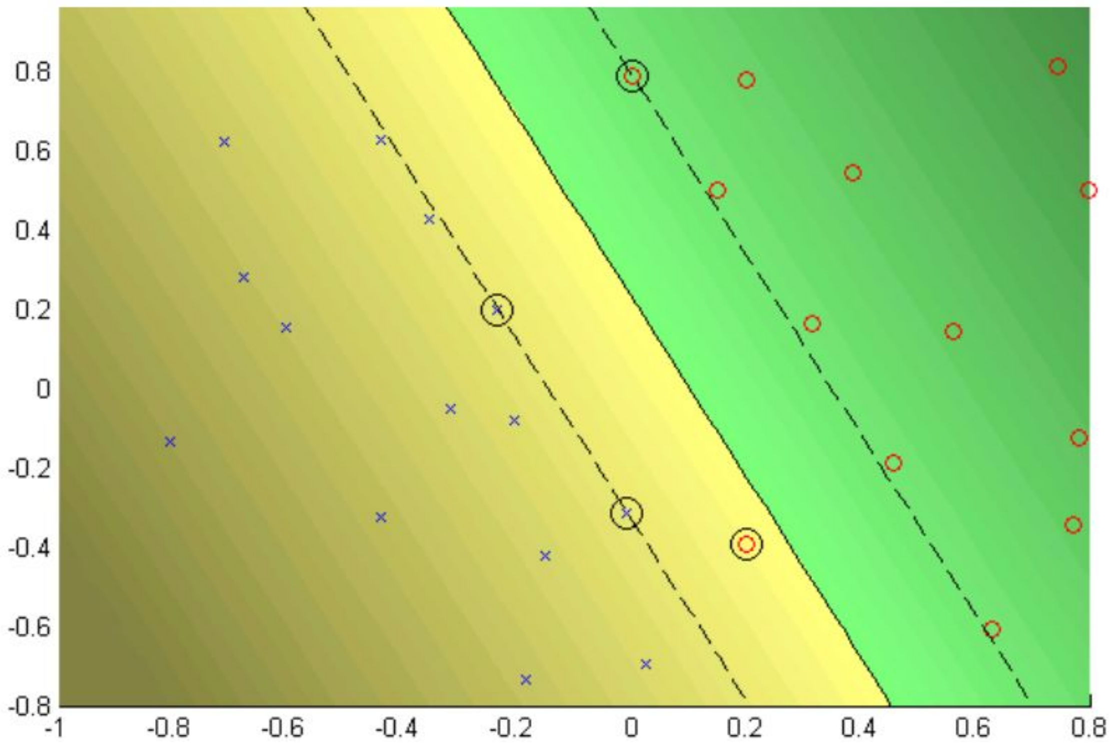
Soft Margin

- Quanto maior o número da variável de folga (C), mais *outliers* serão descartados.
- ***C*** $\rightarrow \infty$ hard margin



Soft Margin

- Quanto maior o número da variável de folga (C), mais *outliers* serão descartados.
- ***C*** = 10   soft margin



Mapeamento não Linear

- A grande maioria dos problemas reais não são linearmente separáveis.
- A pergunta então é:
  - “Como resolver problemas que não são linearmente separáveis com um classificador linear?”

Projetar os dados em um espaço onde os dados são linearmente separáveis.

x_i

$\phi(x_i)$

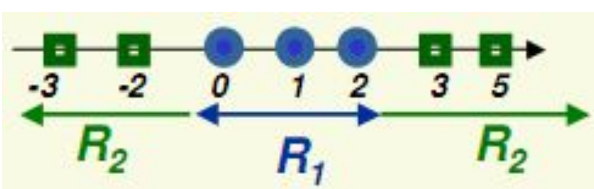
Espaço de entrada

Espaço de características

Mapeamento não Linear

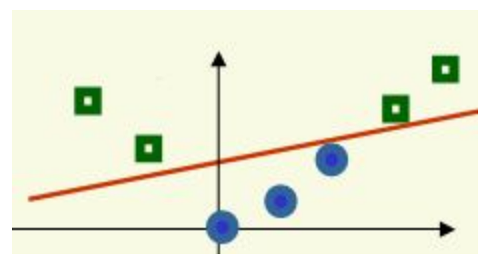
- Projetar os dados em outra dimensão usando uma função de kernel (*kernel trick*).
- Encontrar um hiperplano que separe os dados nesse espaço.

Em qual dimensão esses dados seriam linearmente separáveis?



1D

$$\phi(x, x^2)$$



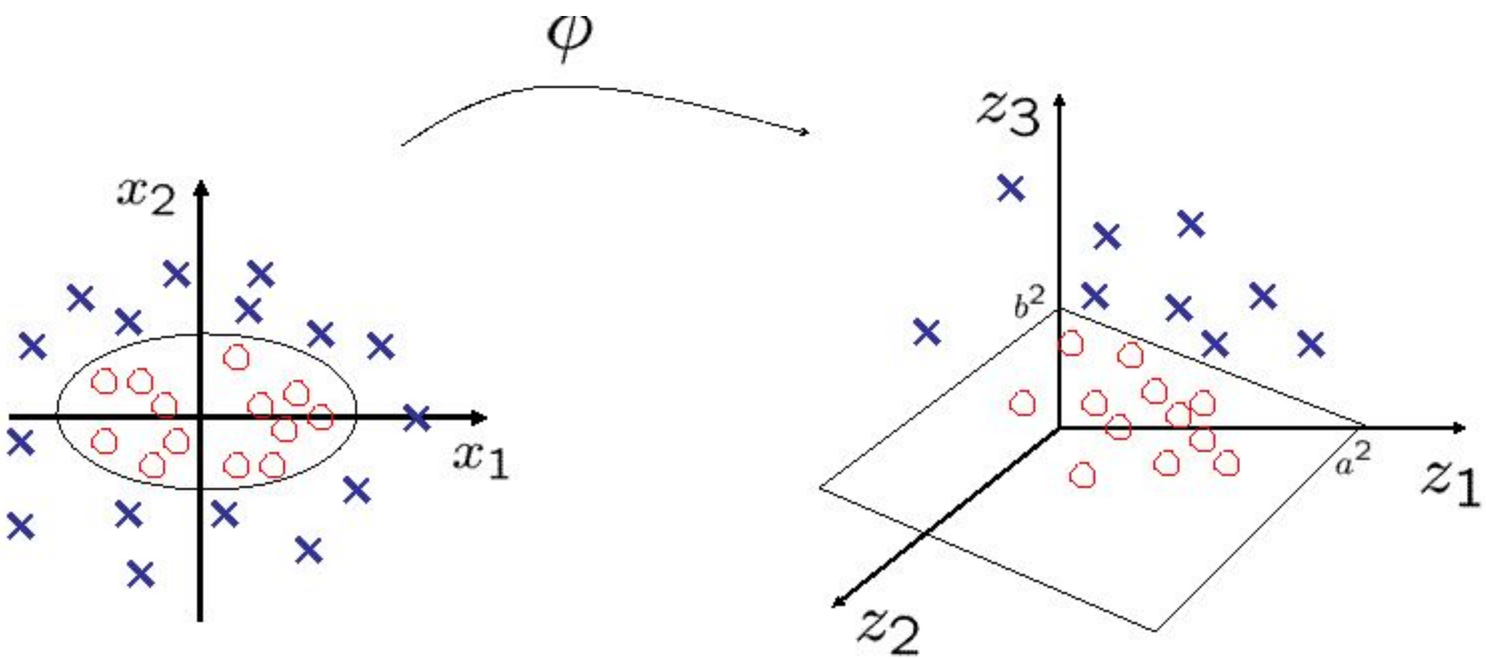
2D

Kernel Trick

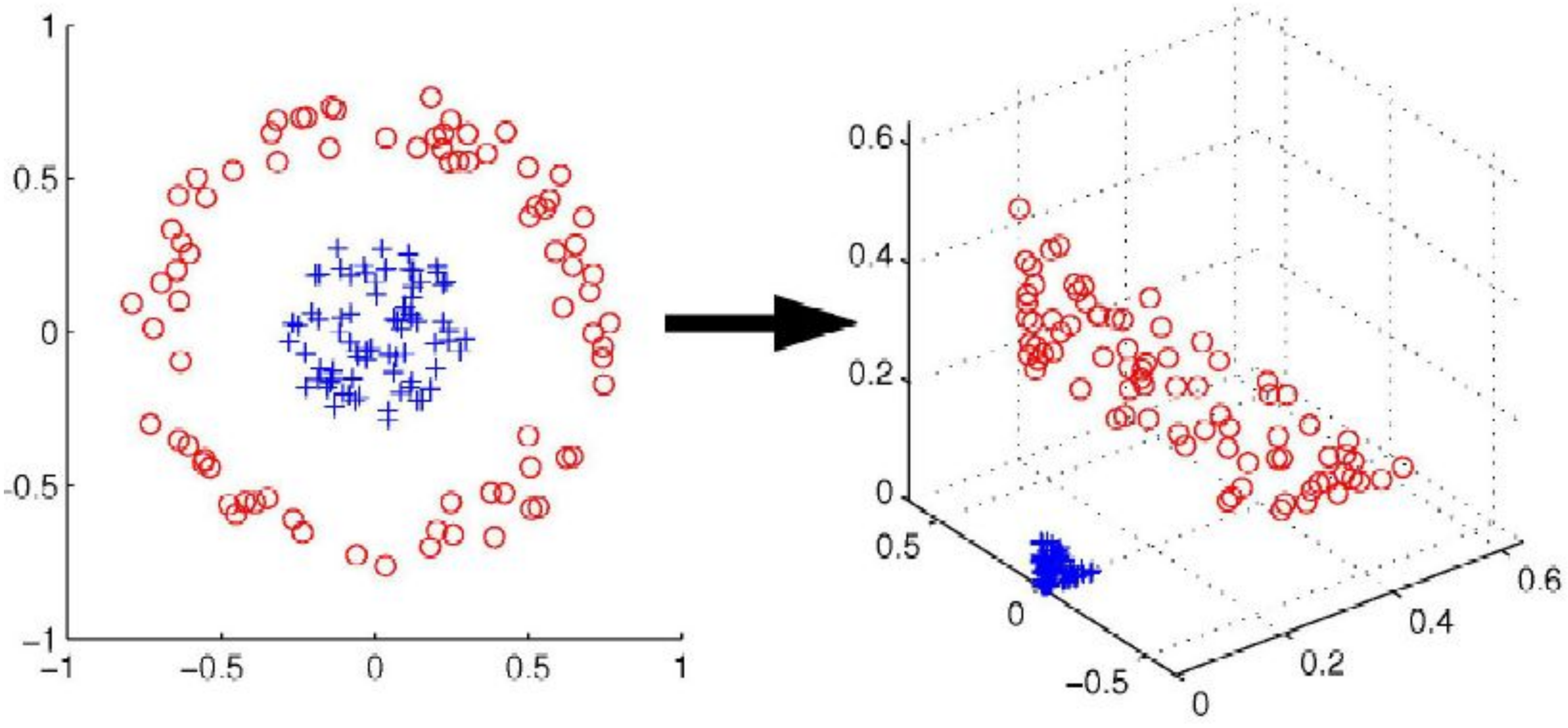
- A função que projeta o espaço de entrada no espaço de características é conhecida como *Kernel*
- Baseado no teorema de Cover
  - Dados no espaço de entrada são transformados (transf. não linear) para o espaço de características, onde são linearmente separáveis.
- O vetor $\phi(x_i)$ representa a “imagem” induzida no espaço de características pelo vetor de entrada

Exemplo

$$\begin{aligned} \phi : \quad \Re^2 &\longrightarrow \Re^3 \\ (x_1, x_2) &\longmapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \end{aligned}$$



Exemplo



Kernel Trick

- Permite construir um hiperplano no espaço de característica sem ter que considerar o próprio espaço de características de forma explícita.
- Toda vez que um produto interno entre vetores deve ser calculado, utiliza-se o kernel.
- Uma função de kernel deve satisfazer o teorema de Mercer (1909) para ser válida.
  - Função definida positiva simétrica

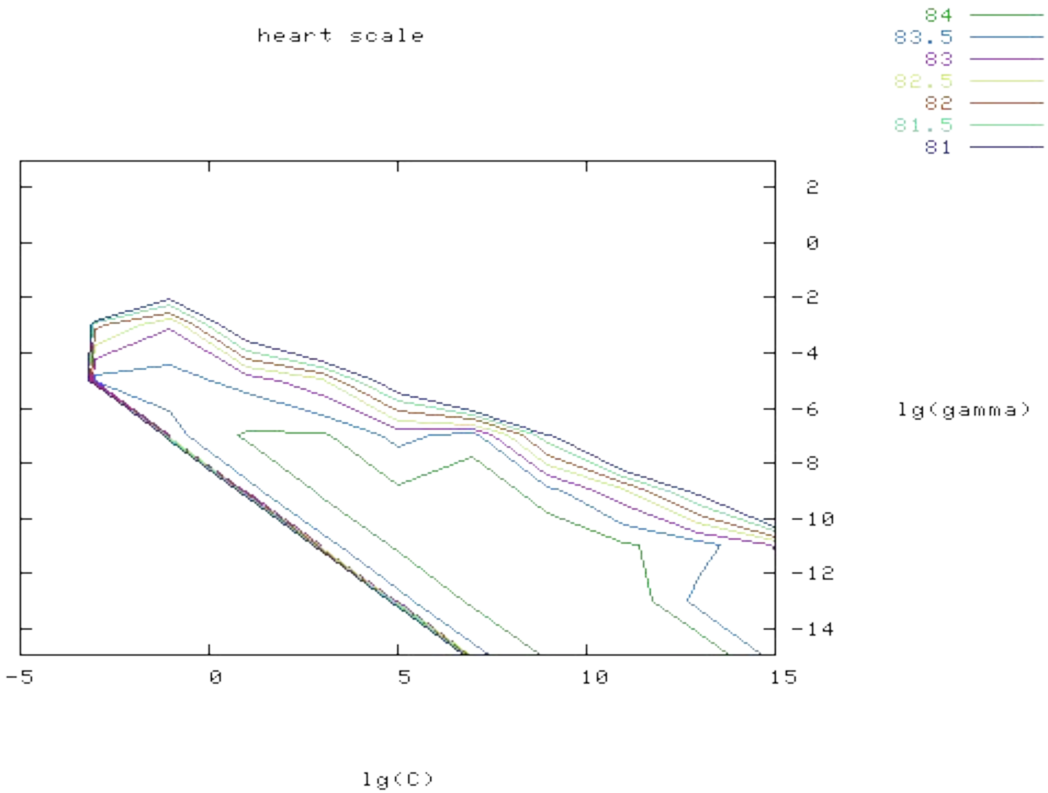
Exemplos de Kernel

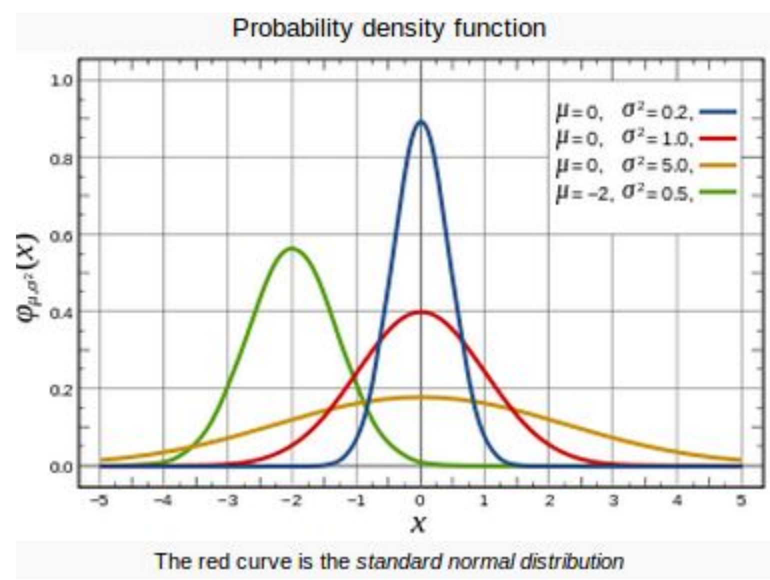
Kernel	Inner Product Kernel
Linear	$K(x,y) = (x \cdot y)$
Gaussian	$K(x,y) = \exp\left(-\frac{\|x-x_i\|^2}{2\sigma^2}\right)$
Polynomial	$K(x,y) = (x \cdot y)^p$
Tangent Hyperbolic	$K(x,y) = \tanh(x \cdot y - \Theta)$

- *Kernels* complexos
  - Mais parâmetros

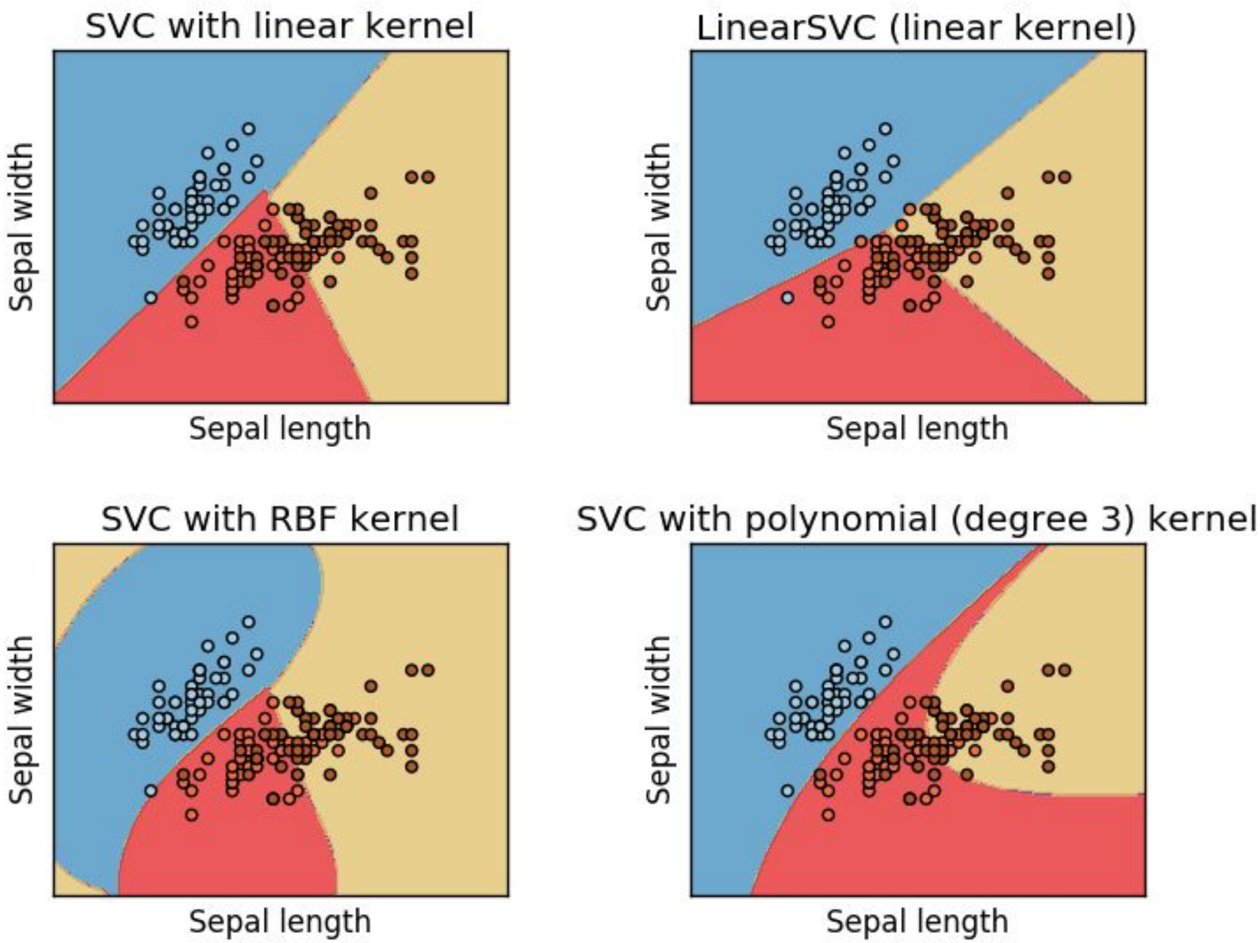
Exemplos de Kernel

- *Grid Search & RBF (Gaussian)*
  - *Gamma & C* (*Margin*)





Exemplos de Kernel

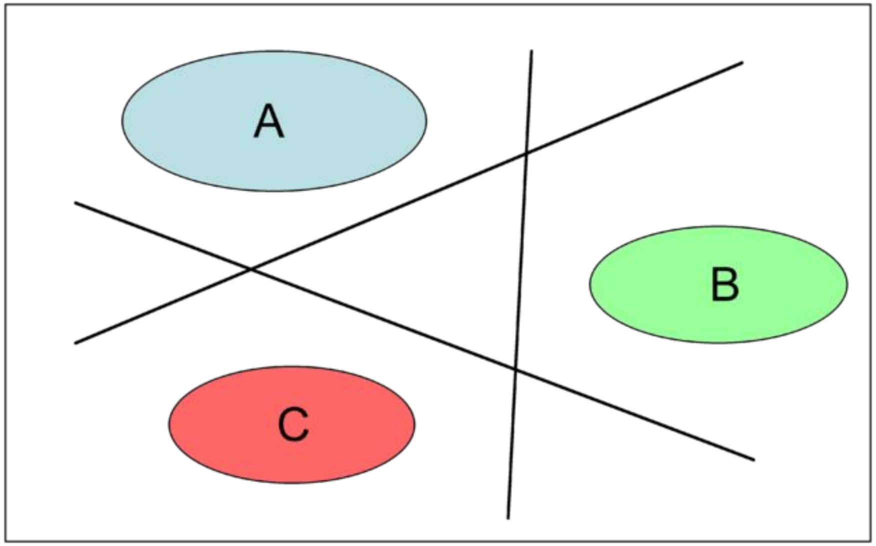


Tomada de Decisão

- SVM são classificadores binários, ou seja, separam duas classes.
- Entretanto, a grande maioria dos problemas reais possuem mais que duas classes.
- Como utilizar os SVMs nesses casos?
  - Um-contra-todos
  - Pairwise

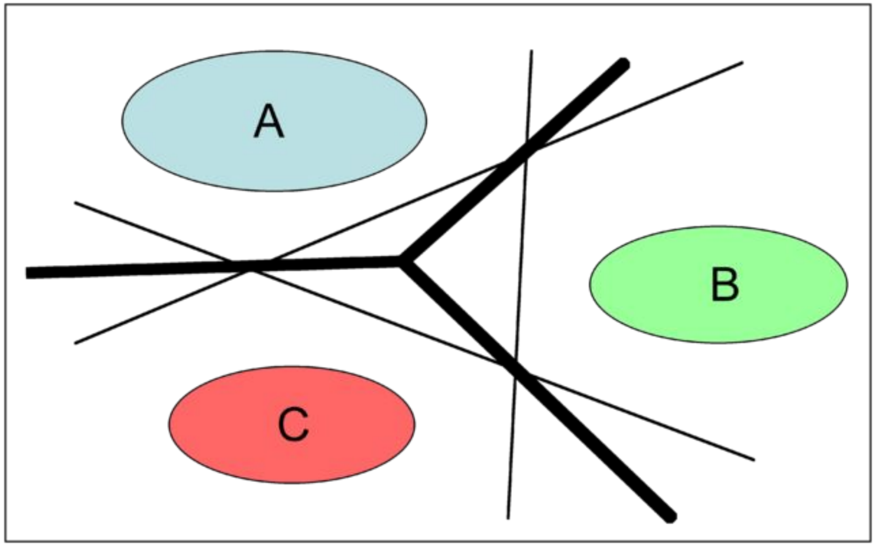
Um-Contra-Todos

- Aqui, o número de classificadores é igual a q.
- Treina-se um classificador c_i para a primeira classe, usando-se como contra exemplos as outras classes, e assim por diante.
- Para se obter a decisão final pode-se utilizar uma estratégia de **votos**.



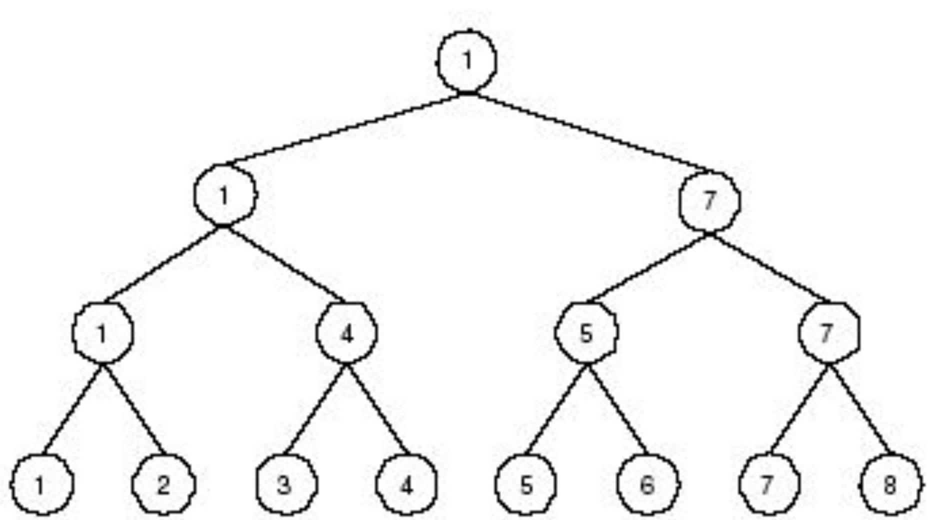
Um-Contra-Todos

- Aqui, o número de classificadores é igual a q.
- Treina-se um classificador c_i para a primeira classe, usando-se como contra exemplos as outras classes, e assim por diante.
- Para se obter a decisão final pode-se utilizar uma estratégia de **votos**.



Pairwise

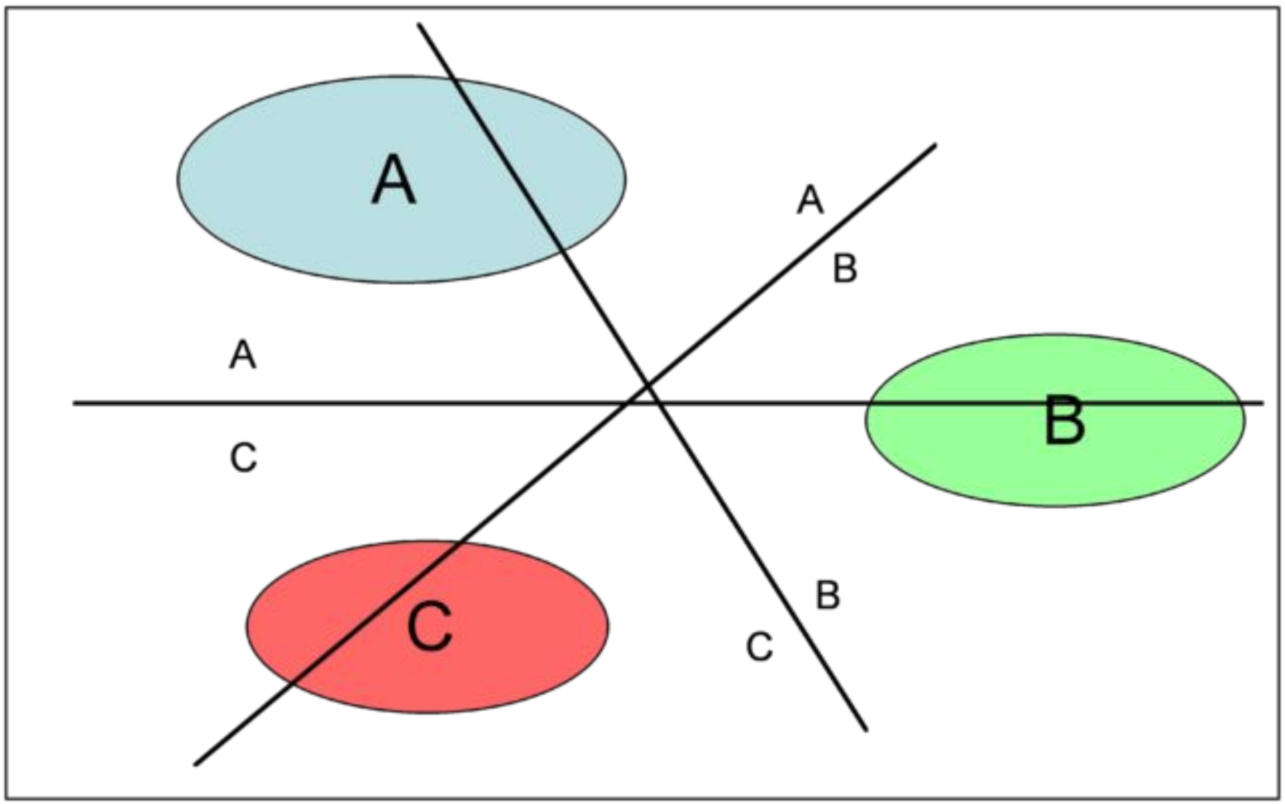
- Consiste em treinar classificadores *pairwise* e arranjá-los em uma árvore



A competição se dá nos níveis inferiores, e o ganhador chegará ao nó principal da árvore.

Número de classificadores para q classes = $q(q-1)/2$.

Pairwise



Referências

- Luiz E. S Oliviera, **Classificadores Lineares**, DInf / UFPR, 2014.

- Victor Lavrenko, Introductory Applied Machine Learning, University of Edinburgo, 2014.