Introdução ao Aprendizado de Máquina

Lucas Gonçalves de Moura Leite

Resumo de ontem

- Aprendizado de máquina
  - Supervisionado x Não Supervisionado
- Classificar frutas com base nas suas características
- K-NN
- Treino x Teste (Capacidade de generalização)

Aula de hoje

- Como outros métodos supervisionados aprendem a partir dos dados
- Vantagens e desvantagens de cada um destes métodos
- Como utilizar estes métodos no scikit-learn
- Princípios gerais de aprendizado de máquina (overfitting)

Nomenclatura

- Representação por atributos (features)
- Instâncias (exemplos)
- Saídas desejadas(target, labels)

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79
5	2	mandarin	mandarin	80	5.8	4.3	0.77
6	2	mandarin	mandarin	80	5.9	4.3	0.81
7	2	mandarin	mandarin	76	5.8	4.0	0.81
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89
10	1	apple	braeburn	166	6.9	7.3	0.93
11	1	apple	braeburn	172	7.1	7.6	0.92
12	1	apple	braeburn	154	7.0	7.1	0.88
13	1	apple	golden_delicious	164	7.3	7.7	0.70
14	1	apple	golden_delicious	152	7.6	7.3	0.69
15	1	apple	golden_delicious	156	7.7	7.1	0.69
16	1	apple	golden_delicious	156	7.6	7.5	0.67
17	1	apple	golden_delicious	168	7.5	7.6	0.73
18	1	apple	cripps_pink	162	7.5	7.1	0.83

Revisão

- Conjuntos de treinamento e testes
- Treinamento (Estimação)
- Avaliação

```
%matplotlib notebook
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

fruits = pd.read_table('fruit_data_with_colors.txt')

X = fruits[['height', 'width', 'mass', 'color_score']]
y = fruits['fruit_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
print("Accuracy of K-NN classifier on test set: ", knn.score(X_test, y_test))

example_fruit = [[5.5, 2.2, 10, 0.70]]
print("Predicted fruit type for ", example_fruit, " is ", knn.predict(example_fruit))
```

Aprendizado Supervisionado

- Classificação e Regressão
  - Saídas discretas ou contínuas
- Muitos métodos supervisionados possuem versões para classificação e regressão

Aprendizado Supervisionado

- Classificação e Regressão
  - Saídas discretas ou contínuas
- Muitos métodos supervisionados possuem versões para classificação e regressão
- Classificação
  - K-NN
- Regressão
  - Regressão linear usando mínimos quadrados

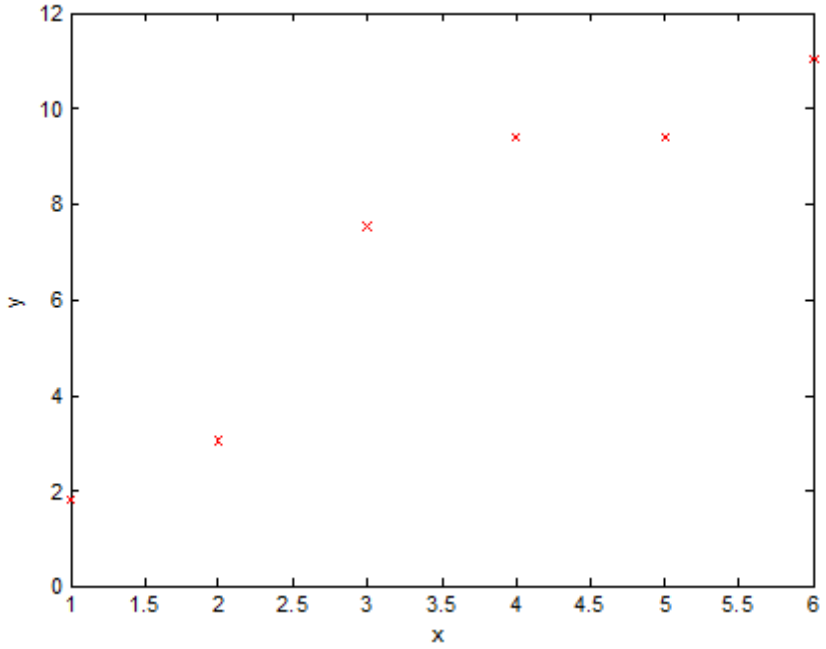
Métodos Supervisionados

- Veremos alguns dos principais métodos de AS
  - Como funcionam (alto nível)
  - Principais hiper-parâmetros
  - Vantagens e desvantagens

Overfitting

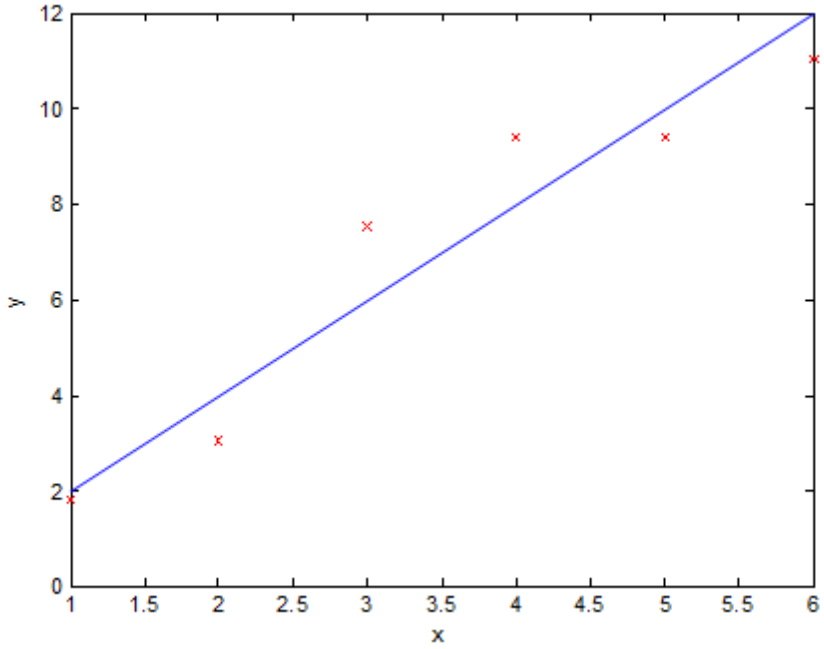
Overfitting

- Exemplo



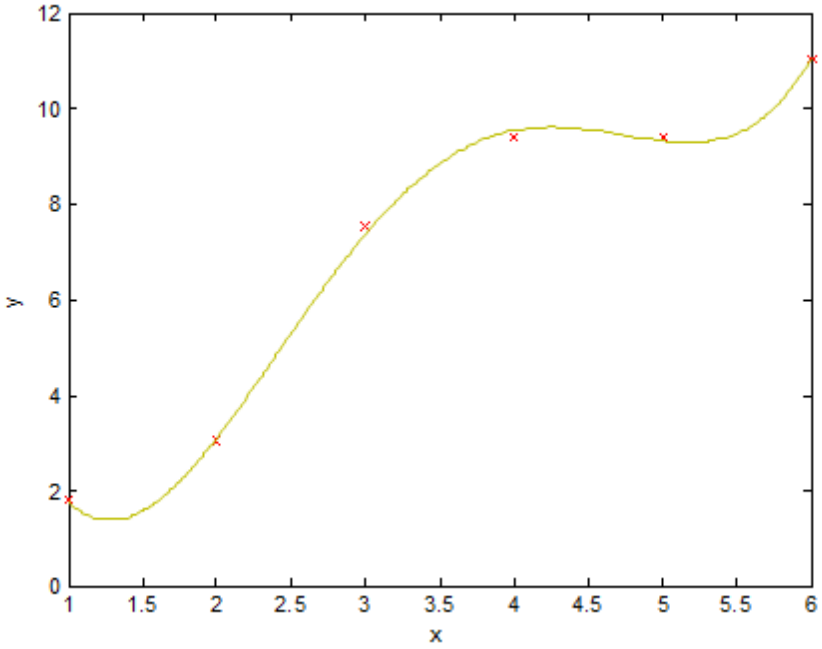
Overfitting

- Exemplo
  - Erro = 6.54



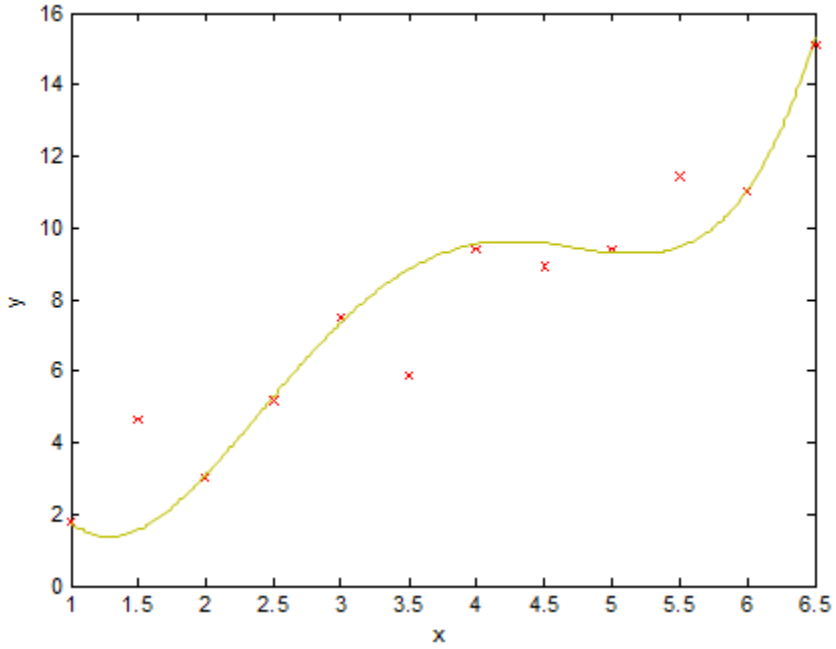
Overfitting

- Exemplo
  - Erro = 2.24



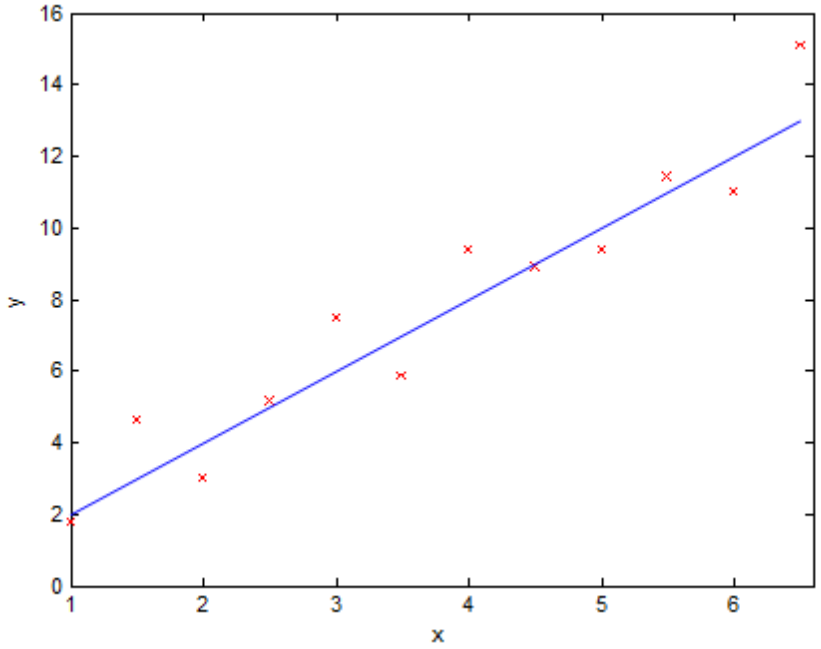
Overfitting

- Exemplo
  - Erro = 27.77



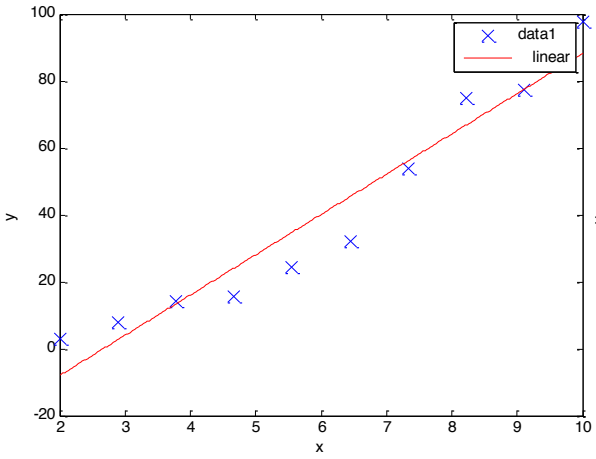
Overfitting

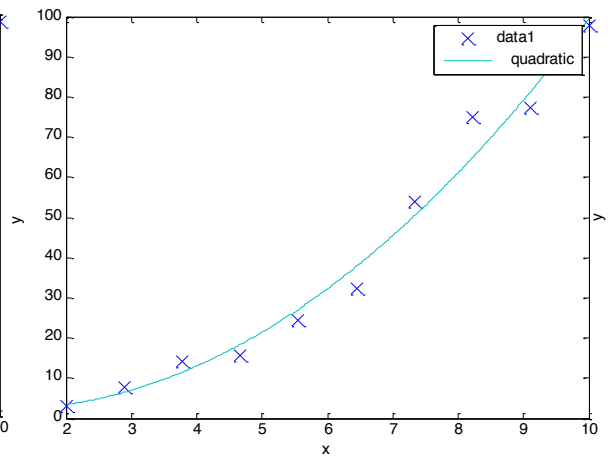
- Exemplo
  - Erro = 15.18

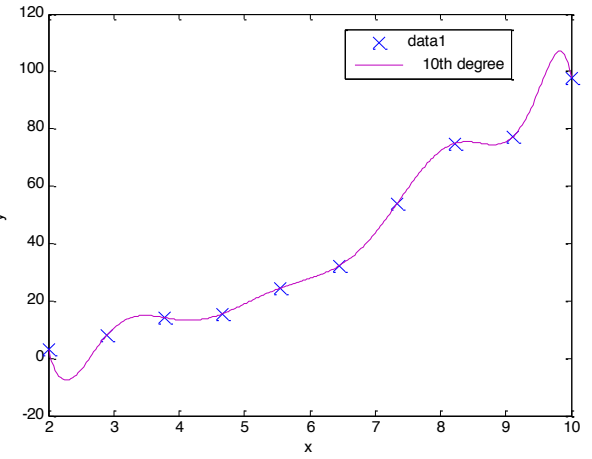


Overfitting

- Modelo se ajusta demasiadamente aos dados utilizados para encontrar os parâmetros

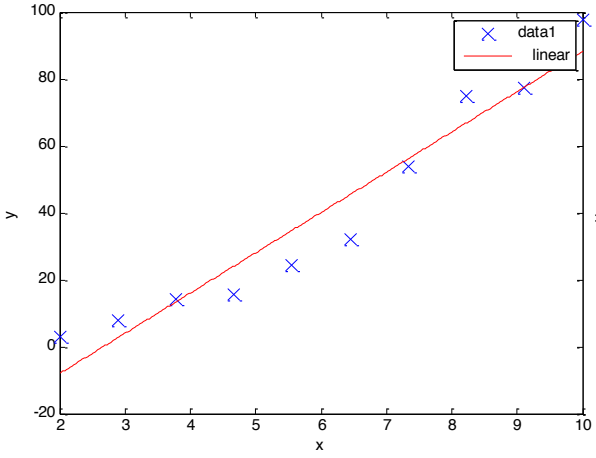




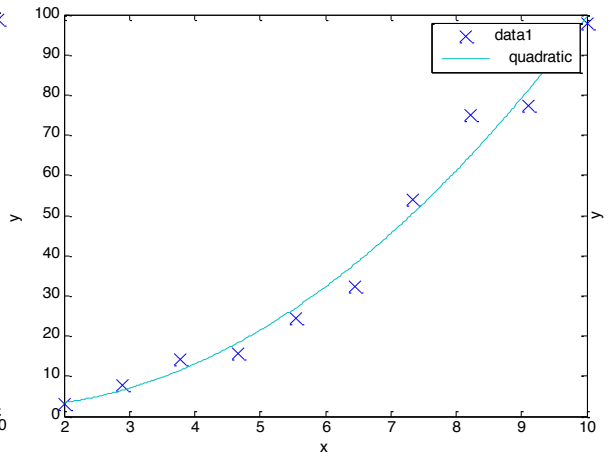


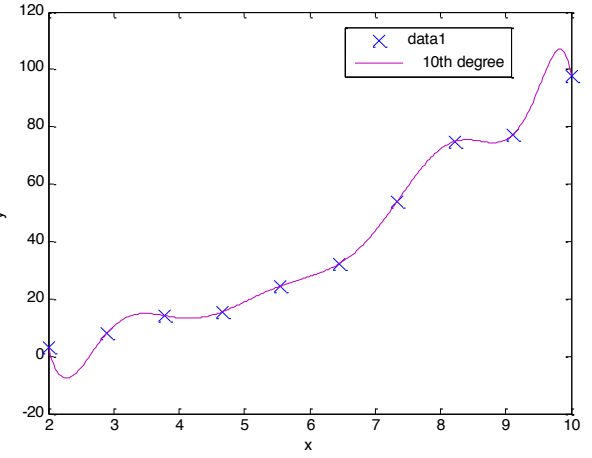
Overfitting

- Modelo se ajusta demasiadamente aos dados utilizados para encontrar os parâmetros



Underfit



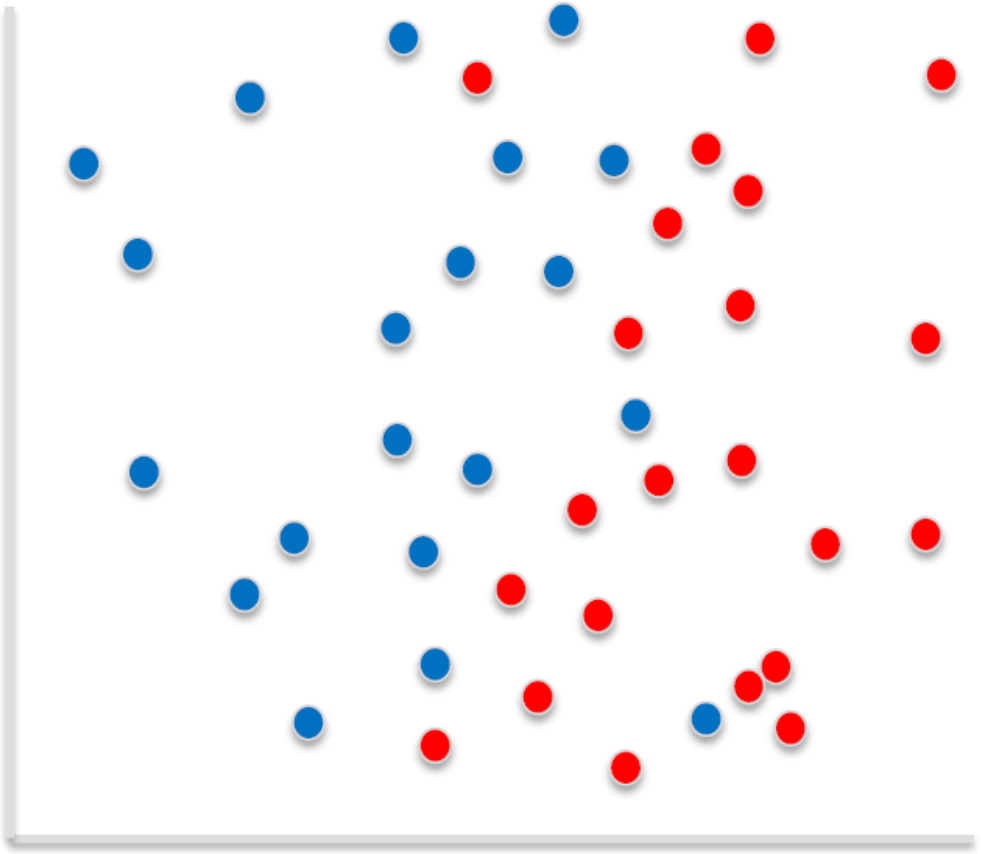


Overfit

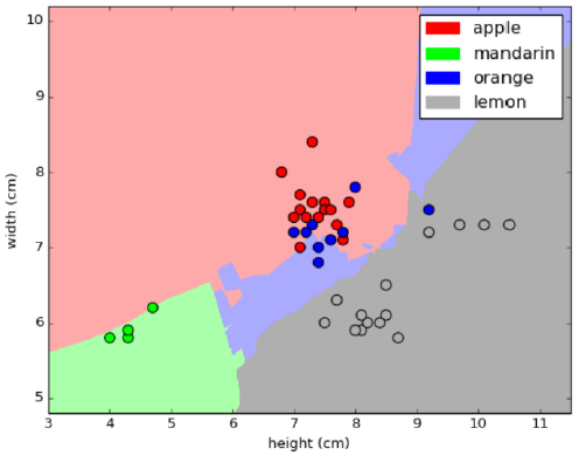
Generalização e Overfitting

- Generalização é a capacidade de um algoritmo de fornecer boas predições para um dado ainda não visto.
- Premissas
  - Dados que não foram vistos se assemelham aos dados usados para treinamento
  - Com isso, modelos ajustados para o treino ficarão bons para o teste
- Overfitting
  - Acontece quando um modelo complexo é ajustado em um conjunto de dados que não é suficientemente grande

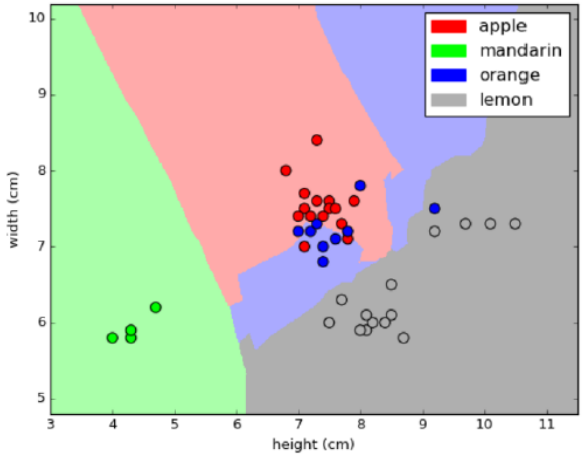
Overfitting para Classificação



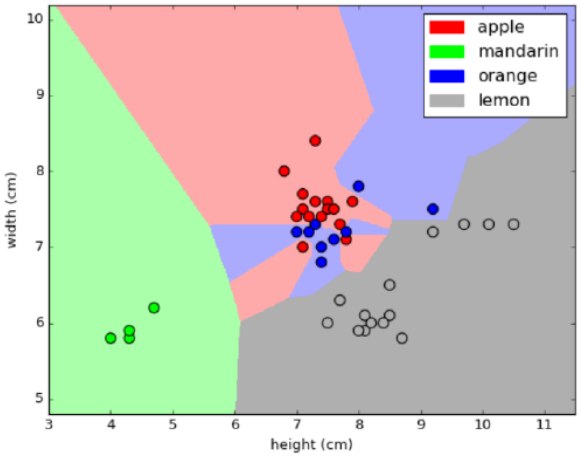
Overfitting K-NN



K=10

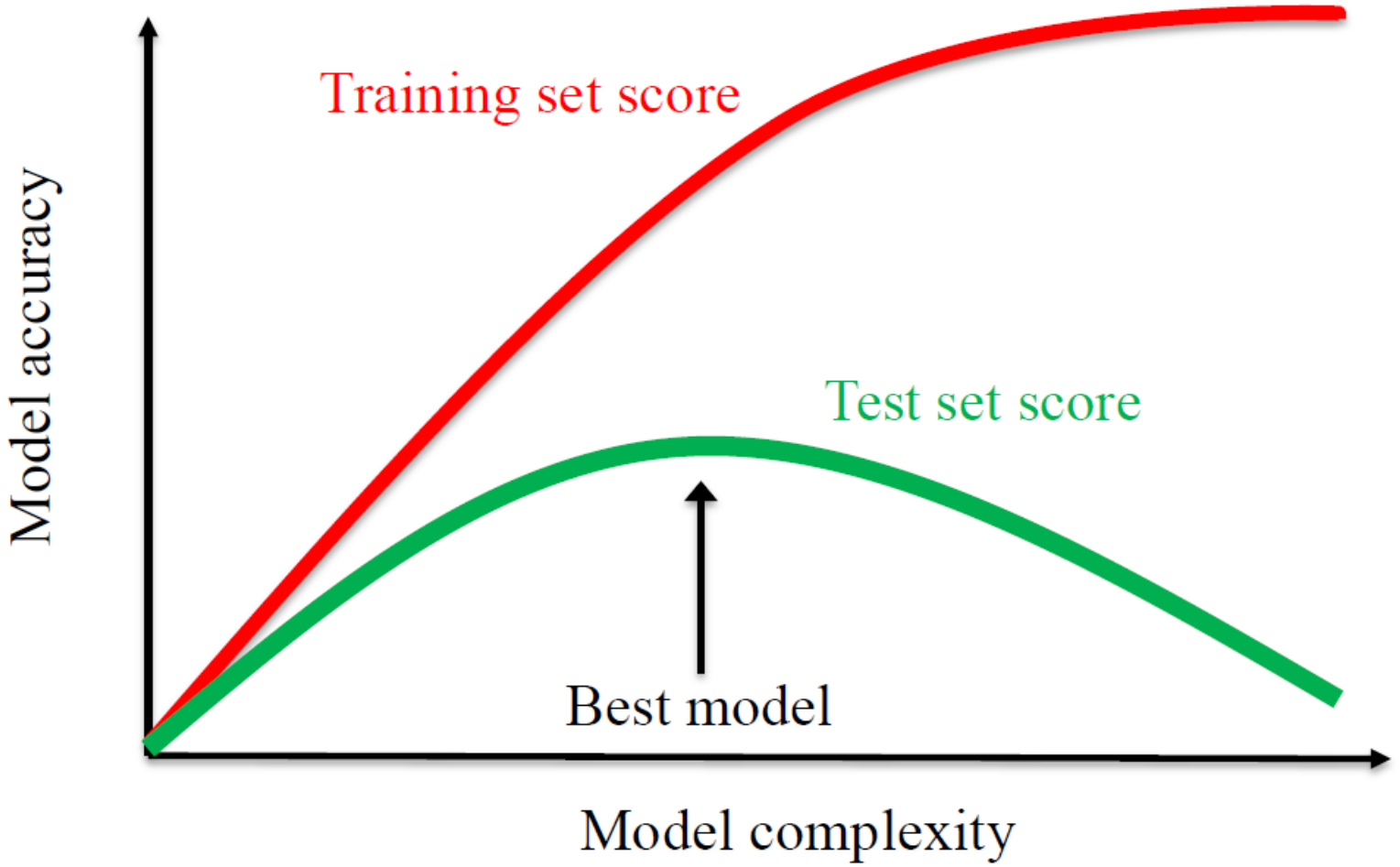


K=5



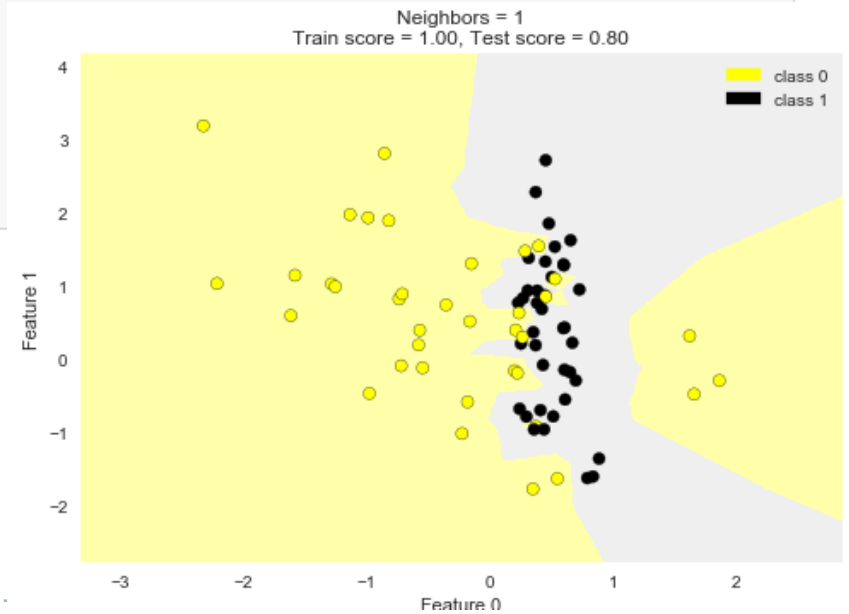
K=1

Comportamento Geral

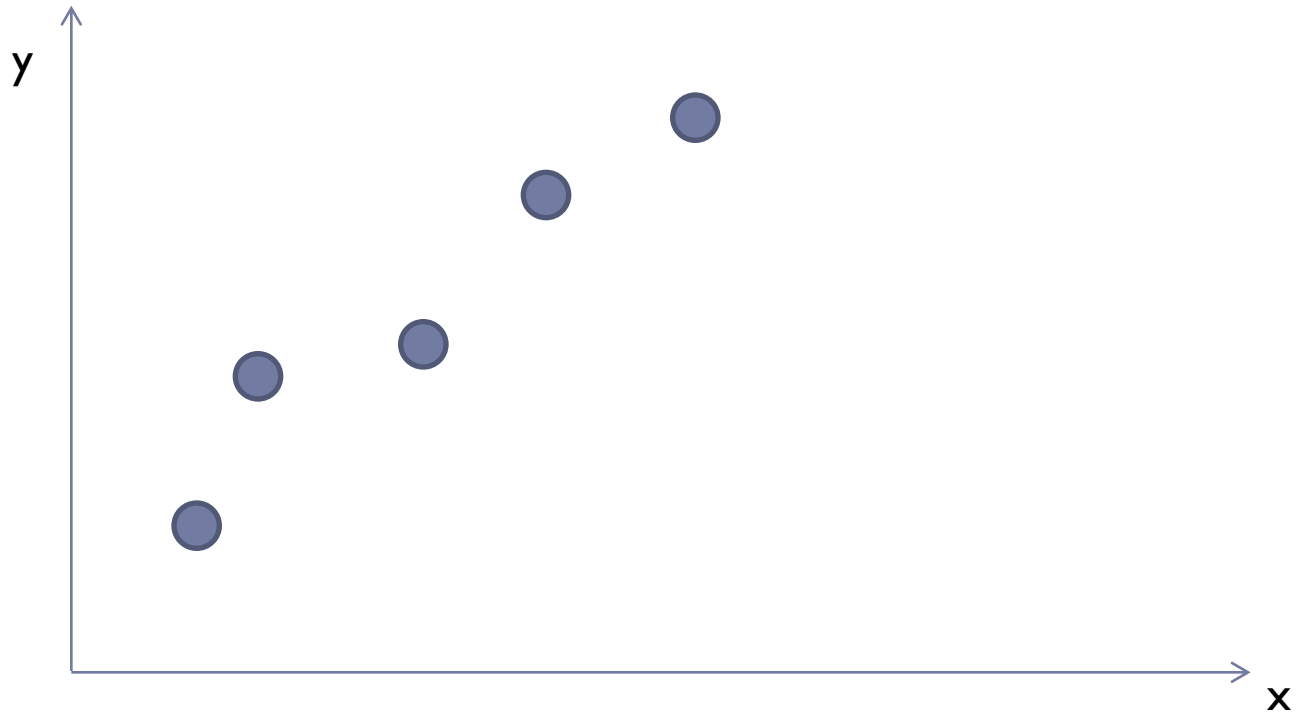


K-NN

```
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import make_classification, make_blobs
from matplotlib.colors import ListedColormap
from adspy_shared_utilities import plot_two_class_knn


df = pd.read_csv('dadosClassBin.csv',index_col=0)

X_C2 = np.array(df[['x1','x2']])
y_C2 = np.array(df['y'])

X_train, X_test, y_train, y_test = train_test_split(X_C2, y_C2,
                                                    random_state=0)
plot_two_class_knn(X_train, y_train, 1, 'uniform', X_test, y_test)
plot_two_class_knn(X_train, y_train, 3, 'uniform', X_test, y_test)
plot_two_class_knn(X_train, y_train, 11, 'uniform', X_test, y_test)
```

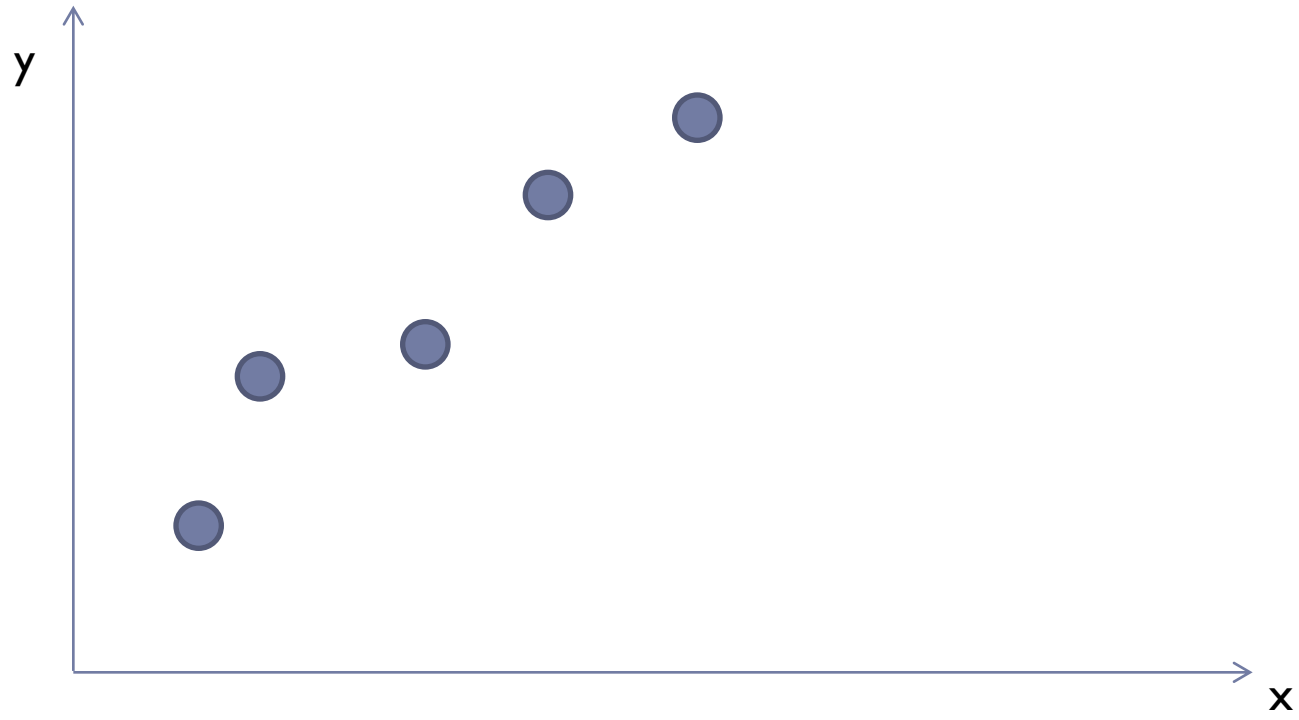


K-NN para Regressão

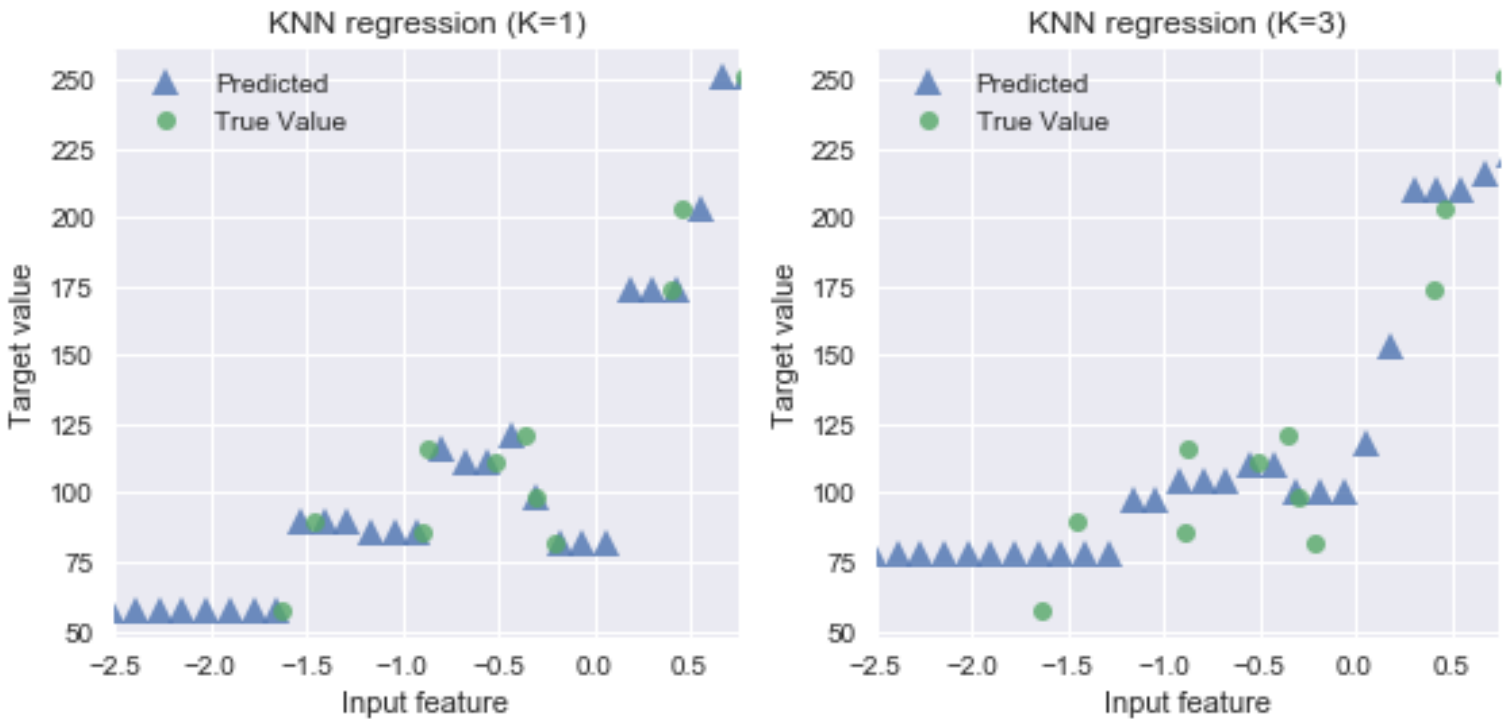


K-NN para Regressão

- Encontra-se os k vizinhos mais próximos
- $\bar{y}$ é dado pela média dos valores de y dos k vizinhos mais próximos



K-NN para Regressão

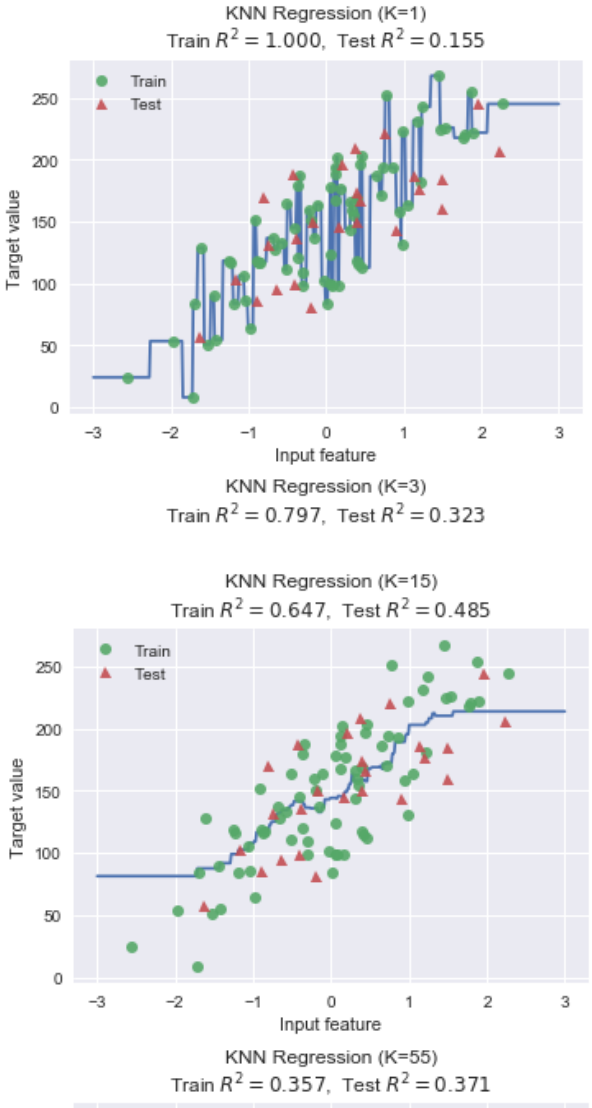


R^2 score

- Mede o quão bem um modelo de regressão se ajustou aos dados
- Entre 0 e 1
  - 1 indica ajuste perfeito

Regressão com variação de K

```
# plot k-NN regression on sample dataset for different values of K
fig, subaxes = plt.subplots(5, 1, figsize=(5,20))
X_predict_input = np.linspace(-3, 3, 500).reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X_R1, y_R1,
                                                    random_state = 0)

for thisaxis, K in zip(subaxes, [1, 3, 7, 15, 55]):
    knnreg = KNeighborsRegressor(n_neighbors = K).fit(X_train, y_train)
    y_predict_output = knnreg.predict(X_predict_input)
    train_score = knnreg.score(X_train, y_train)
    test_score = knnreg.score(X_test, y_test)
    thisaxis.plot(X_predict_input, y_predict_output)
    thisaxis.plot(X_train, y_train, 'o', alpha=0.9, label='Train')
    thisaxis.plot(X_test, y_test, '^', alpha=0.9, label='Test')
    thisaxis.set_xlabel('Input feature')
    thisaxis.set_ylabel('Target value')
    thisaxis.set_title('KNN Regression (K={})\n\
Train $R^2 = {:.3f}$,  Test $R^2 = {:.3f}$'
                       .format(K, train_score, test_score))
    thisaxis.legend()
    plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)
```



Modelos lineares para regressão

Regressão Linear

- Explicar uma determinada variável de saída (y) através de variáveis de entrada (x), utilizando um modelo linear.
- Exemplo
  - Concessão de crédito

Regressão Linear

- Concessão de Crédito

Salário (R$)	Crédito (R$)
1500	16500
2000	18000
3000	28000
3300	33000
4200	49000
...	...
5500	52000

Regressão Linear

- Concessão de Crédito
  - Encontrar uma relação linear entre Salário e Credito

Salário (R$)	Crédito (R$)
1500	16500
2000	18000
3000	28000
3300	33000
4200	49000
...	...
5500	52000

Regressão Linear

- Concessão de Crédito
  - Encontrar uma relação linear entre Salário e Credito
  - Salário (x_i) e Crédito (y_i)

Salário (R$)	Crédito (R$)
1500	16500
2000	18000
3000	28000
3300	33000
4200	49000
...	...
5500	52000

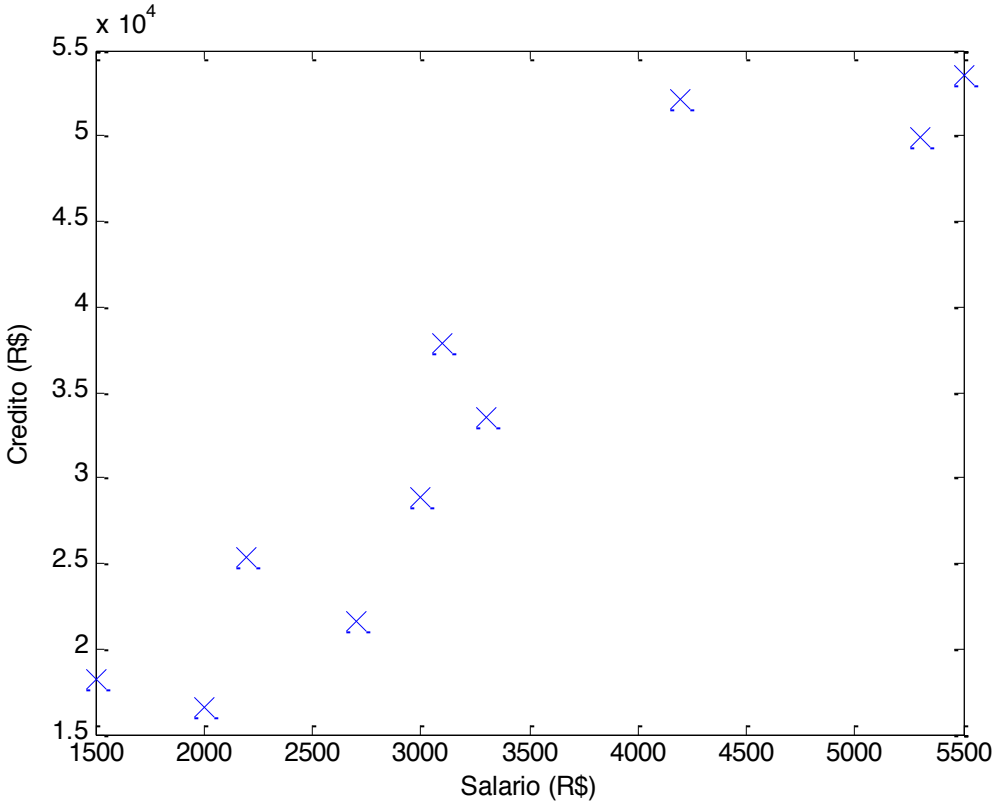
Regressão Linear

- Concessão de Crédito
  - Encontrar uma relação linear entre Salário e Credito
  - Salário (x_i) e Crédito (y_i)
  - $\overline{y_i} = w_1 x_i + w_0$

Salário (R$)	Crédito (R$)
1500	16500
2000	18000
3000	28000
3300	33000
4200	49000
...	...
5500	52000

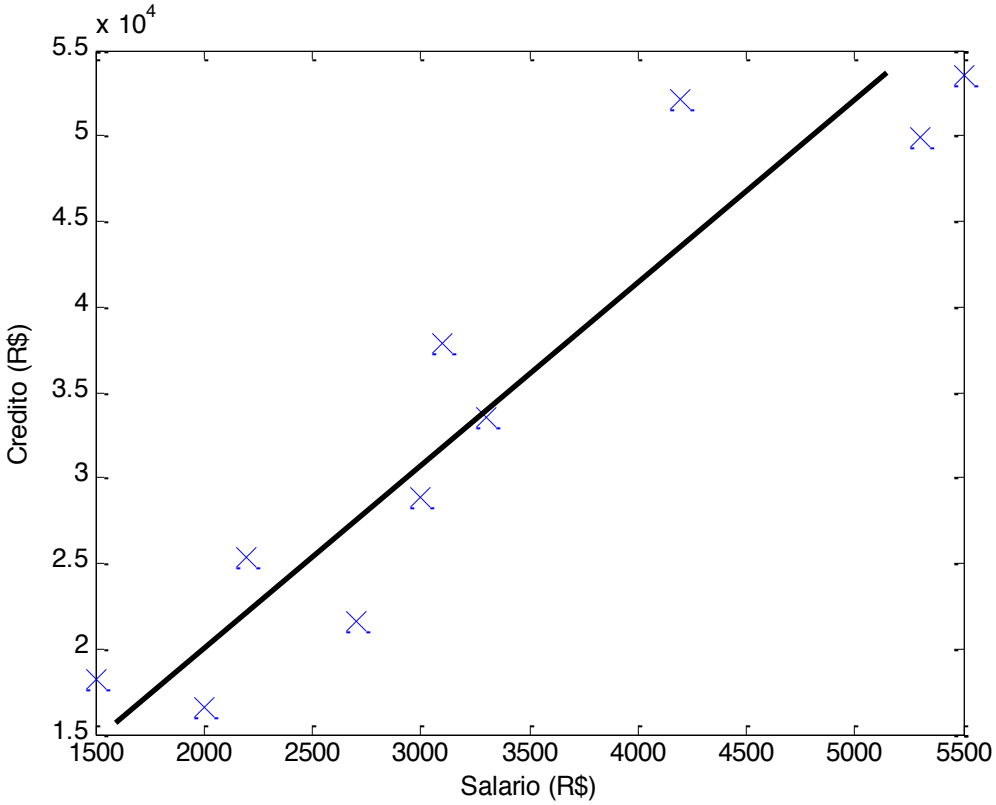
Regressão Linear

- $\overline{y_i} = w_1 x_i + w_0$



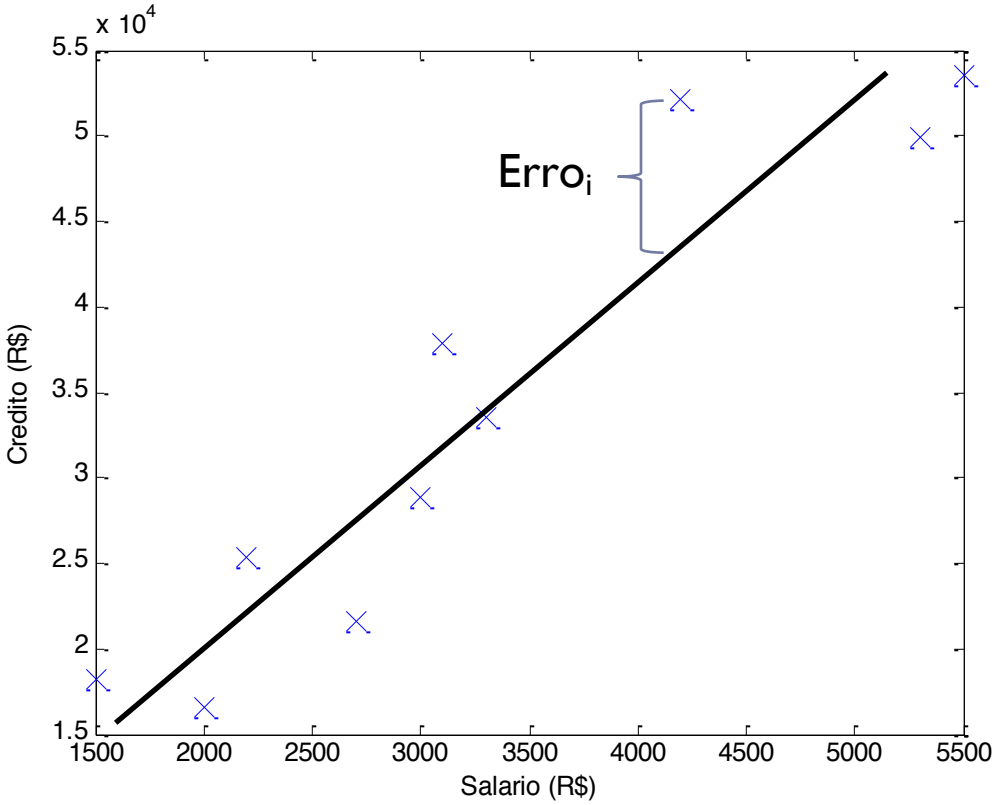
Regressão Linear

- $\overline{y_i} = w_1 x_i + w_0$



Regressão Linear

- $\overline{y_i} = w_1 x_i + w_0$



Regressão Linear

- $\bar{y_i} = w_1 x_i + w_0$
- $J(w_1, w_0) = \frac{1}{2n} \sum_{i=1}^{n} {e_i}^2$
  - Onde: $e_i = y_i - \bar{y_i}$

Regressão Linear

- $\bar{y_i} = w_1 x_i + w_0$
- $J(w_1, w_0) = \frac{1}{2n} \sum_{i=1}^{n} {e_i}^2$
  - Onde: $e_i = y_i - \bar{y_i}$

- $min_{w_1 w_0} J(w_1, w_0)$

Regressão Linear

- $\bar{y}_i = w_1 x_i + w_0$
- $J(w_1, w_0) = \frac{1}{2n} \sum_{i=1}^{n} {e_i}^2$
  - Onde: $e_i = y_i - \bar{y}_i$

- $min_{w_1 w_0} J(w_1, w_0)$

Ordinary Least-Squares (OLS)

Estimação dos Parâmetros

- w_1 e w_0 são estimados a partir dos dados de treinamento
- Existem muitos métodos para este ajuste
  - Minimizar uma função de perda (loss function)

Regressão Linear Multivariada

- Em diversos problemas é necessário utilizar mais de uma variável x para tentar explicar a variável de saída y

Regressão Linear Multivariada

- Em diversos problemas é necessário utilizar mais de uma variável x para tentar explicar a variável de saída y
  - Exemplo
    - Concessão de crédito

Regressão Linear Multivariada

- Em diversos problemas é necessário utilizar mais de uma variável x para tentar explicar a variável de saída y
  - Exemplo
    - Concessão de crédito

Salário (R$)	Dívida (R$)	Crédito (R$)
1500	0	16500
2000	4000	12000
3000	2000	28000
...		...
5500	1700	52000

Regressão Linear Multivariada

- Concessão de Crédito
  - Encontrar uma relação linear entre Salário,Dívida e Credito
  - Salário (x_{1i}), Salário (x_{2i}) e Crédito (y_i)
  - $\overline{y_i} = w_1 x_{1i} + w_2 x_{2i} + w_0$

Salário (R$)	Dívida (R$)	Crédito (R$)
1500	0	16500
2000	4000	12000
3000	2000	28000
...		...
5500	1700	52000

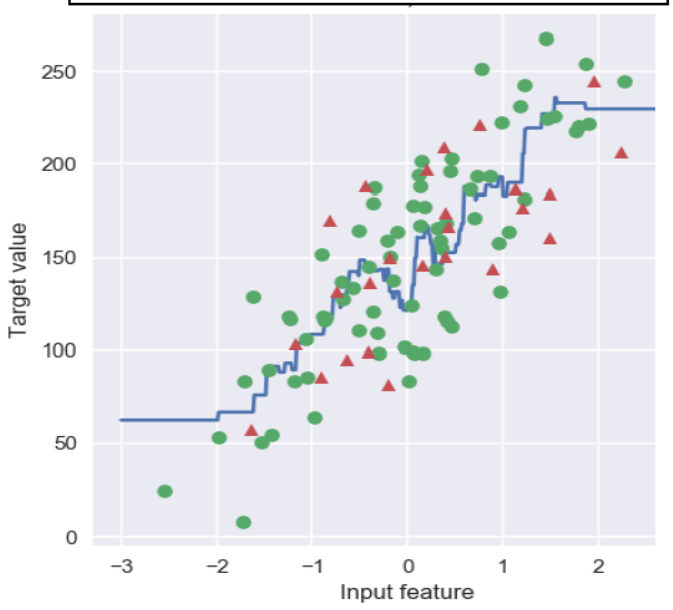
Regressão Linear – Scikit Learn

```
import numpy as np
import pandas as pd
import seaborn as sn
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.datasets import make_classification, make_blobs
from matplotlib.colors import ListedColormap
from sklearn.linear_model import LinearRegression


df = pd.read_csv('dadosRegLin1.csv',index_col=0)

X_R1 = df[['x']]
y_R1 = df['y']


X_train, X_test, y_train, y_test = train_test_split(X_R1, y_R1, random_state = 0)

linreg = LinearRegression().fit(X_train, y_train)
```

KNN x RL

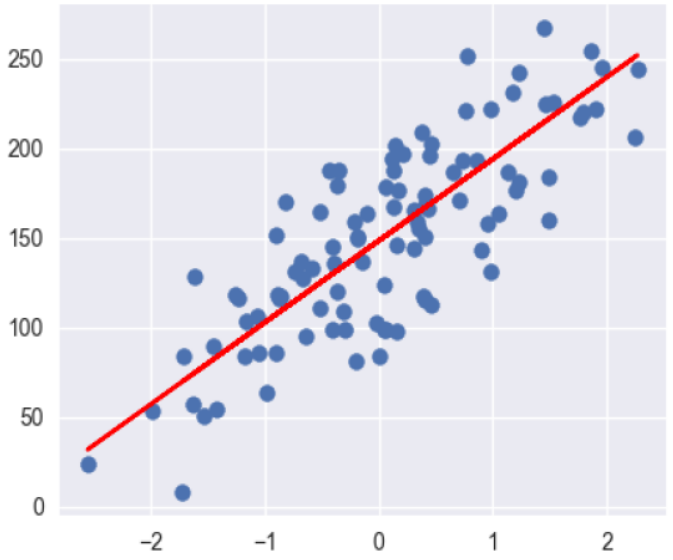
k-NN (k=7)

R-squared score (training): 0.720
R-squared score (test): 0.471



LS linear

R-squared score (training): 0.679
R-squared score (test): 0.492



Ridge Regression

- Regressão Linear
  - Função Objetivo
    - $J(\boldsymbol{w}) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \overline{y_i})^2$
- Regressão Ridge
  - Função Objetivo
    - $J(\boldsymbol{w}) = \frac{1}{2n}[\sum_{i=1}^{n}(y_i - \overline{y_i})^2 + \lambda\sum_{j=1}^{m} {w_j}^2]$

Regressão com regularização L2 (Ridge Regression)

- Função Objetivo
  - $J(\boldsymbol{w}) = \frac{1}{2n}[\sum_{i=1}^{n}(y_i - \bar{y_i})^2 + \lambda \sum_{j=1}^{m} {w_j}^2]$
- Penaliza a utilização dos pesos
- Modelos mais simples
- Menos suscetível a overfitting

Normalização de Dados

- Importante que os atributos estejam na mesma escala
  - K-NN, regressão regularizada
- Normalizações
  - Min-Max
  - Standardization

Normalização Min-Max

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()


X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

- Mesma normalização para treino e teste
  - Não usar dados de teste
- Não normalizar as saídas

Standardization

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(X_train)
scaler.transform(X_test)
```

- Dados com média 0 e variância 1

Regressão Lasso

- Função Objetivo
  - $J(\boldsymbol{w}) = \frac{1}{2n}[\sum_{i=1}^{n}(y_i - \bar{y_i})^2 + \lambda \sum_{j=1}^{m} |w_j|]$
- Produz modelos esparsos
  - Seleção de atributos

```
from sklearn.linear_model import Lasso
from sklearn.preprocessing import MinMaxScaler
from adspy_shared_utilities import load_crime_dataset

(X_crime, y_crime) = load_crime_dataset()


scaler = MinMaxScaler()


X_train, X_test, y_train, y_test = train_test_split(X_crime, y_crime,
                                                    random_state = 0)

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

linlasso = Lasso(alpha=2.0, max_iter = 10000).fit(X_train_scaled, y_train)
```

Modelos não-Lineares com Regressão Linear

- É possível criar regressões não lineares através da formulação linear do problema.
  - Relação entre as variáveis é reconhecidamente não linear (quadrática, cúbica ...)

Modelos não-Lineares com Regressão Linear

- É possível criar regressões não lineares através da formulação linear do problema
- Criar novos atributos

Modelos não-Lineares com Regressão Linear

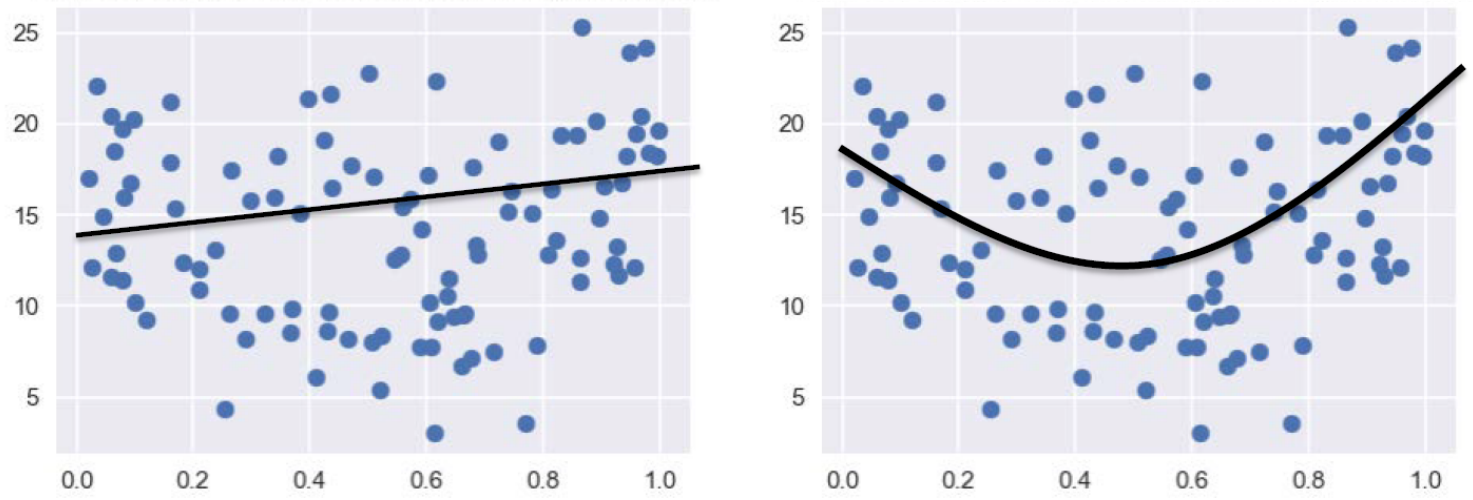
- É possível criar regressões não lineares através da formulação linear do problema
- Criar novos atributos

$$\boldsymbol{x} = (x_0, x_1) \Rightarrow \boldsymbol{x}' = (x_0, x_1, x_0^2, x_0 x_1, x_1^2)$$

$$\hat{y} = \hat{w}_0 x_0 + \hat{w}_1 x_1 + \hat{w}_{00} x_0^2 + \hat{w}_{01} x_0 x_1 + \hat{w}_{11} x_1^2 + b$$

Regressão Polinomial

- Linear nos parâmetros
- Ajuste de dados não lineares




```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2)
X_F1_poly = poly.fit_transform(X_F1)
```

Exercício

- Carregar os dados do dataset boston housing
- Estimar a saida usando todos os modelos lineares vistos até agora

```
>>> from sklearn.datasets import load_boston
>>> boston = load_boston()
```

Modelos lineares para classificação

Regressão Logística

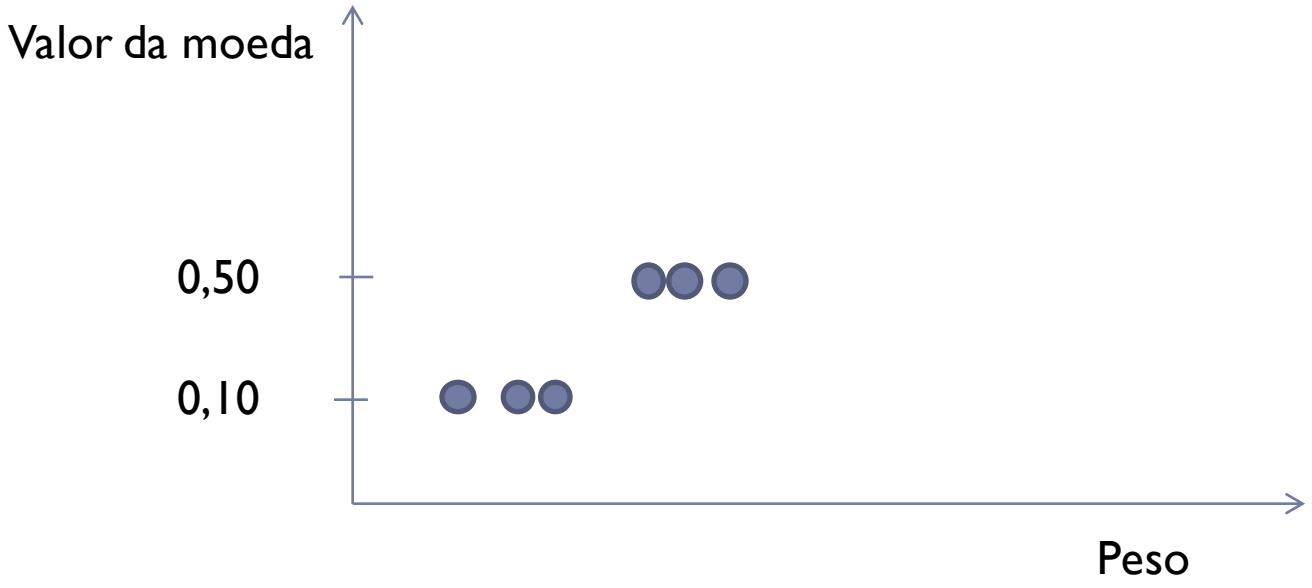
- Método de Classificação

Regressão Logística

- Método de Classificação
  - Exemplo

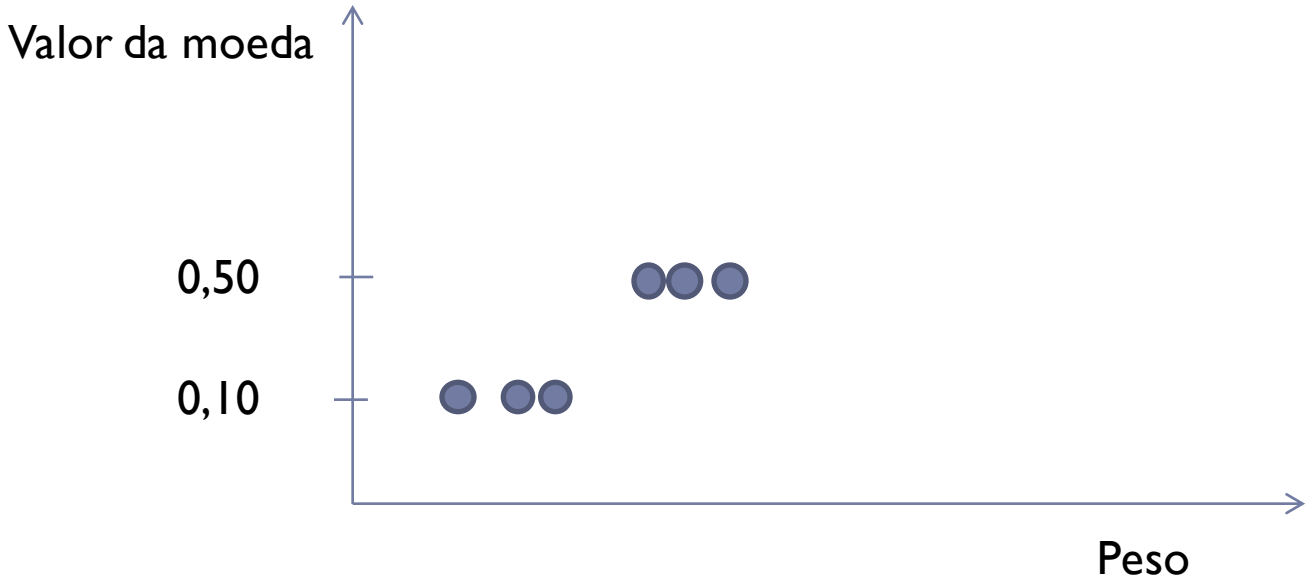
Regressão Logística

- Método de Classificação
  - Exemplo



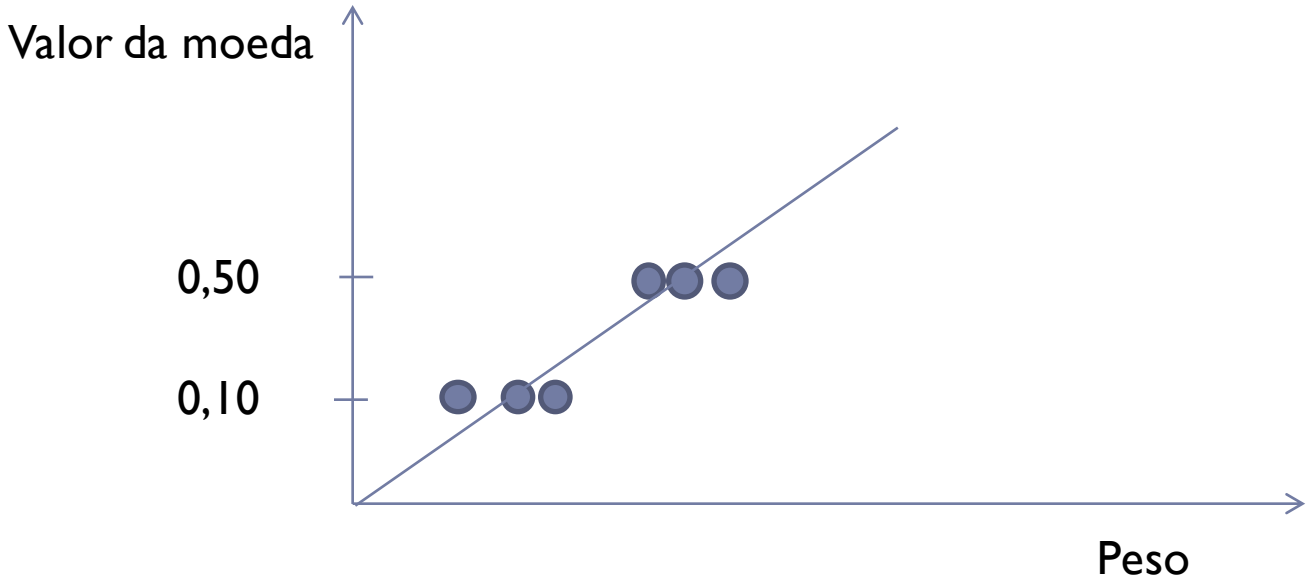
Regressão Logística

- Método de Classificação
  - Exemplo
    - Utilizando Regressão Linear
      - $\bar{y_i} = w_1 x_i + w_0$



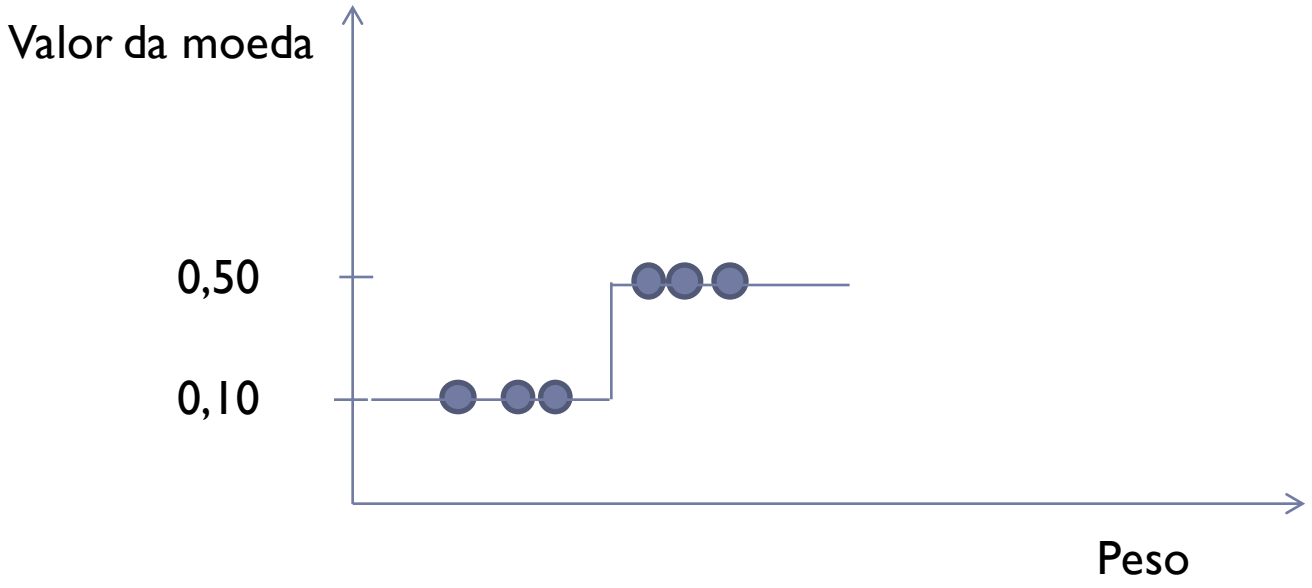
Regressão Logística

- Método de Classificação
  - Exemplo
    - Utilizando Regressão Linear
      - $\overline{y_i} = w_1 x_i + w_0$



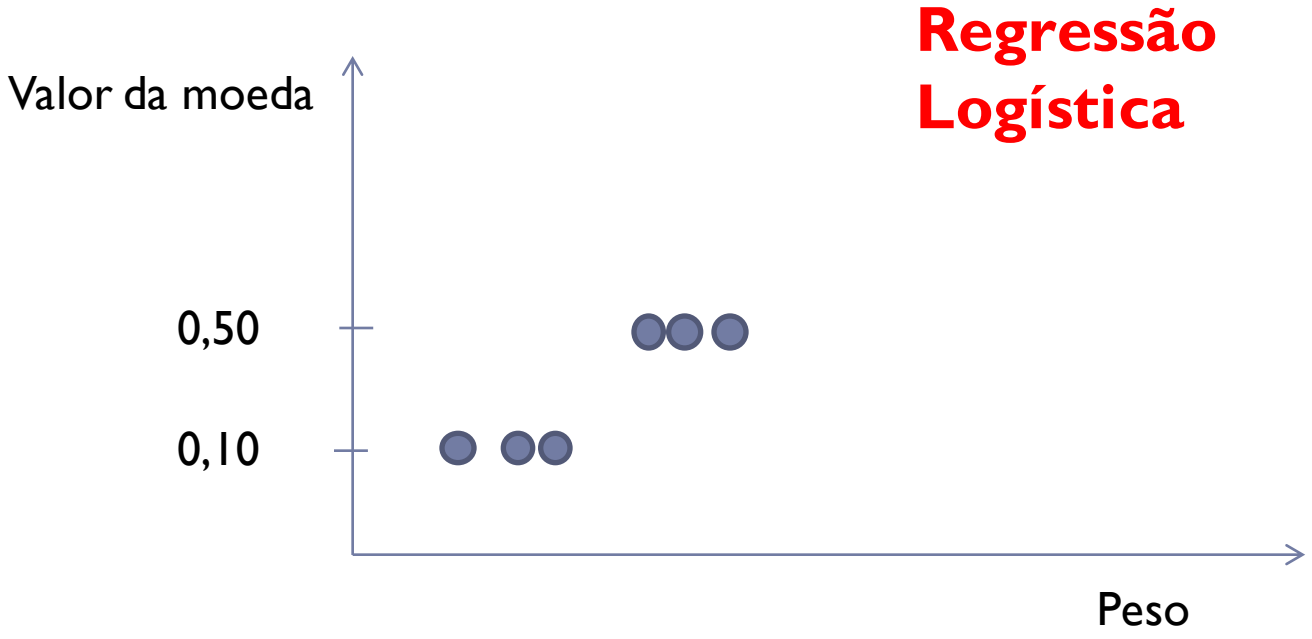
Regressão Logística

- Método de Classificação
  - Exemplo
    - Utilizando Regressão Linear
      - $\overline{y_i} = w_1 x_i + w_0$



Regressão Logística

- Método de Classificação
  - Exemplo
    - Utilizando Regressão Linear
      - $\overline{y_i} = w_1 x_i + w_0$



Regressão Logística

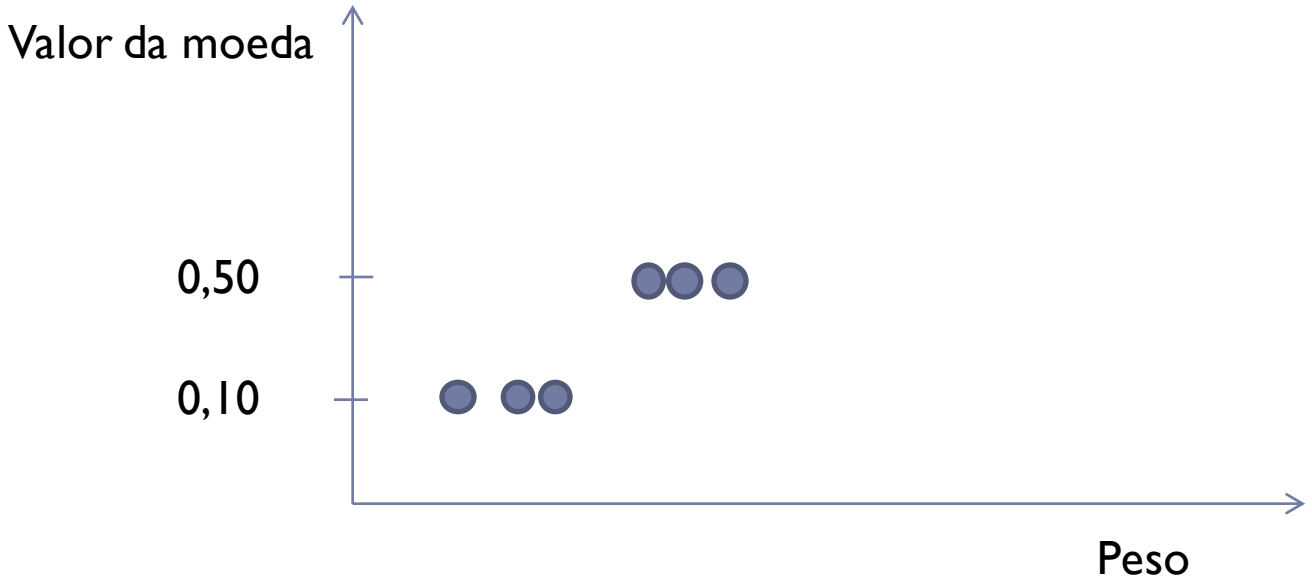
- Modelo Linear
  - $\bar{y_i} = w_1 x_i + w_0 = \boldsymbol{w}^T \boldsymbol{x}_i$
- Função Logística
  - $f(x) = \frac{1}{1+e^{-x}}$

Regressão Logística

- Modelo Linear
  - $\bar{y}_i = w_1 x_i + w_0 = \boldsymbol{w}^T \boldsymbol{x}_i$
- Função Logística
  - $f(x) = \frac{1}{1+e^{-x}}$
- Regressão Logística
  - $\bar{y}_i = \frac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x}_i}}$

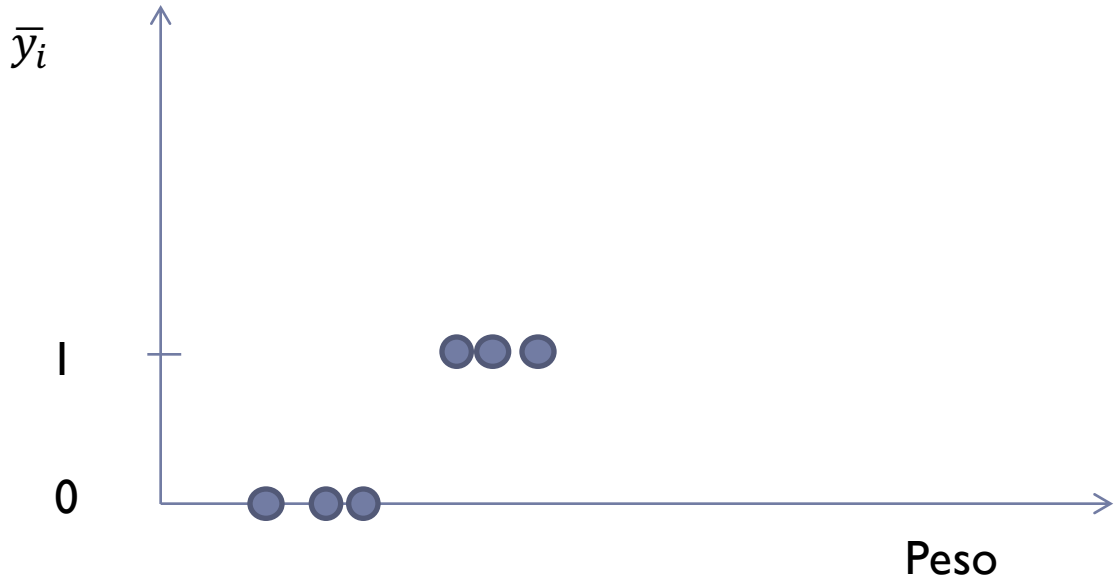
Regressão Logística

- $\overline{y_i} = \frac{1}{1+e^{-w^T x_i}}$



Regressão Logística

- $\overline{y_i} = \frac{1}{1+e^{-w^T x_i}}$



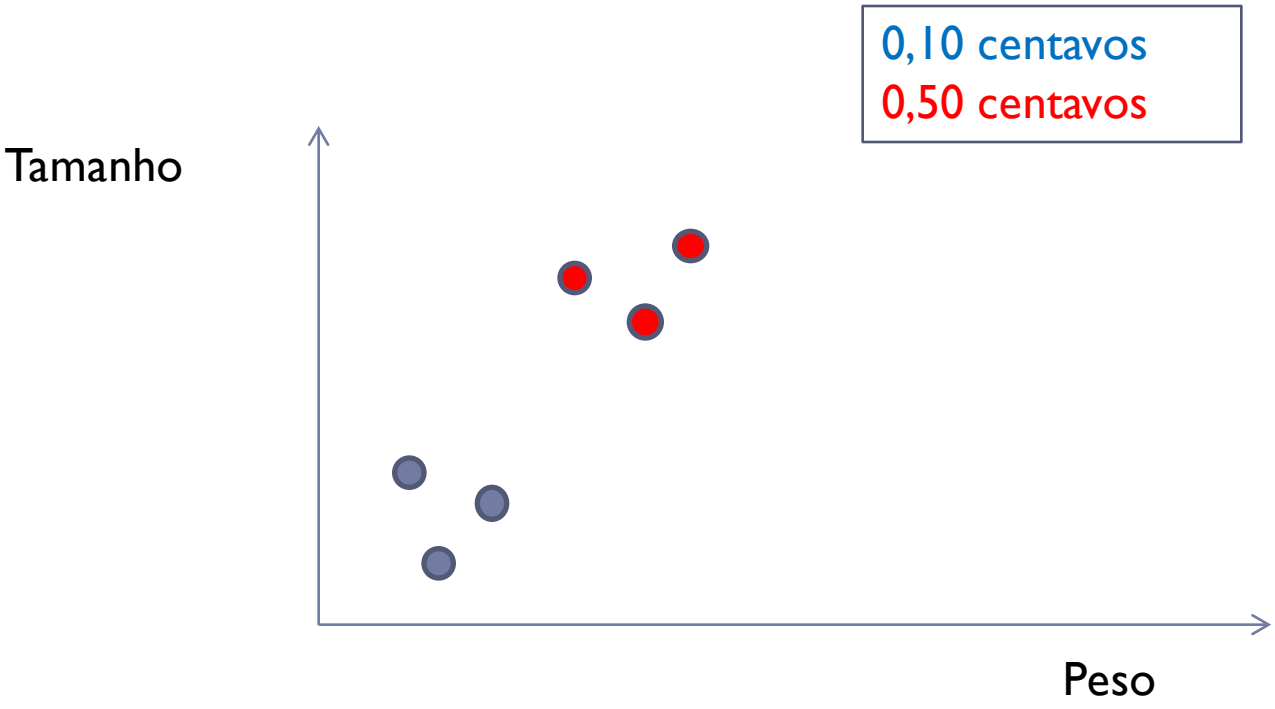
Regressão Logística

- $\bar{y}_i = \frac{1}{1+e^{-w^T x_i}}$

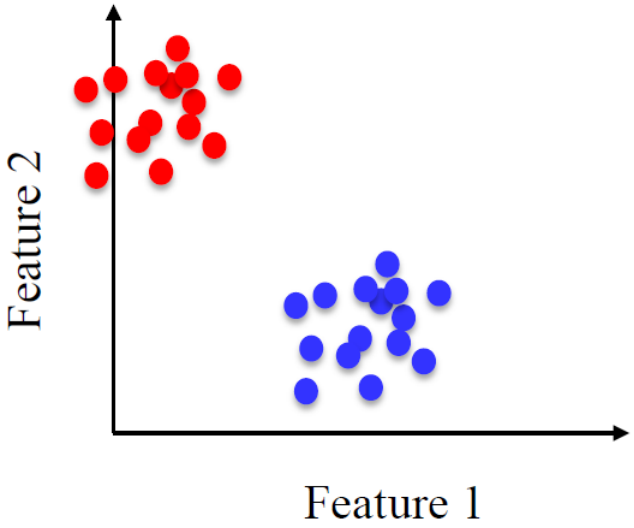
0,10 centavos
0,50 centavos

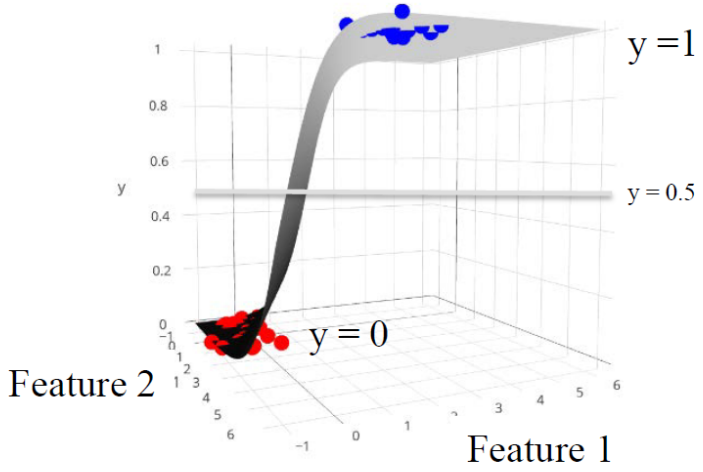
Regressão Logística

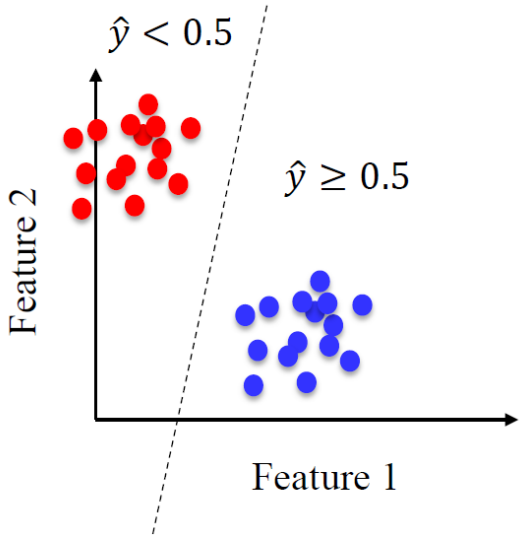
- $\bar{y}_i = \frac{1}{1+e^{-w^T x_i}}$



Regressão Logística







Regressão Logistica

- Regressão Logistica regularizada
  - Função Objetivo
    - $J(\boldsymbol{w}) = \frac{1}{2n}[\sum_{i=1}^{n}(y_i - \overline{y_i})^2 + \lambda \sum_{j=1}^{m} {w_j}^2]$
- Regressão Logistica regularizada (Scikit Learn)
  - Função Objetivo
    - $J(\boldsymbol{w}) = \frac{1}{2n}[\sum_{i=1}^{n}(y_i - \overline{y_i})^2 + \frac{1}{C}\sum_{j=1}^{m} {w_j}^2]$

Regressão Logística

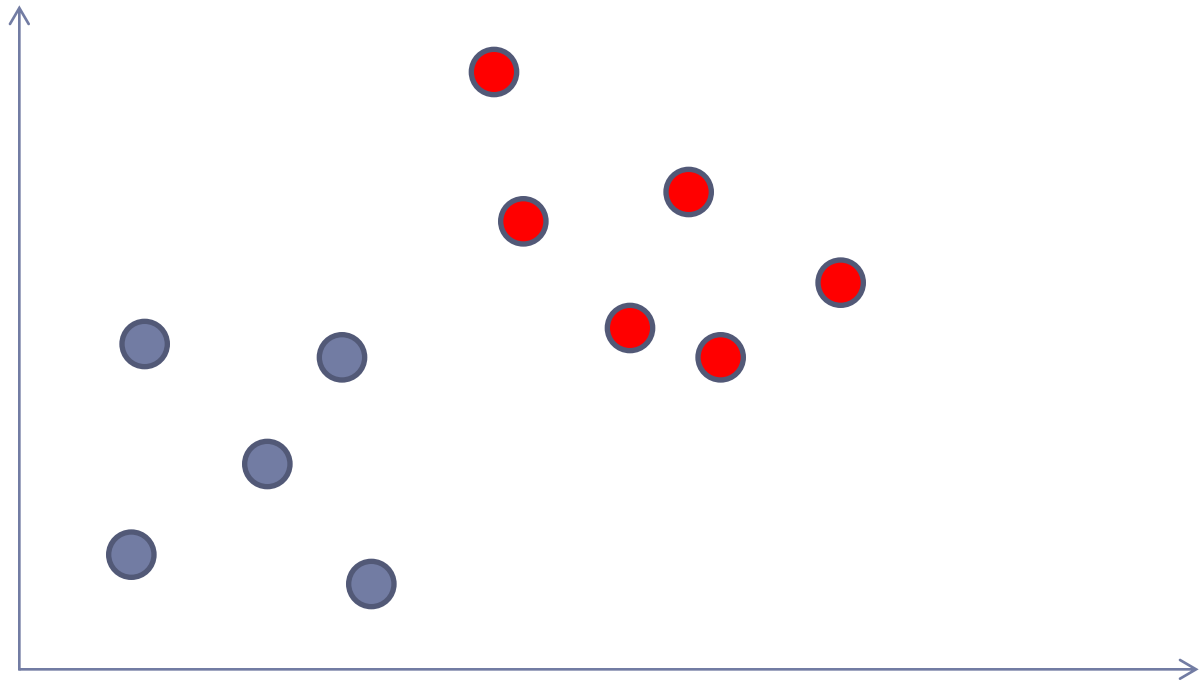
- Parametro de regularização (C)
  - Lógica inversa a da regressão logistica

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=100).fit(X_train, y_train)
```

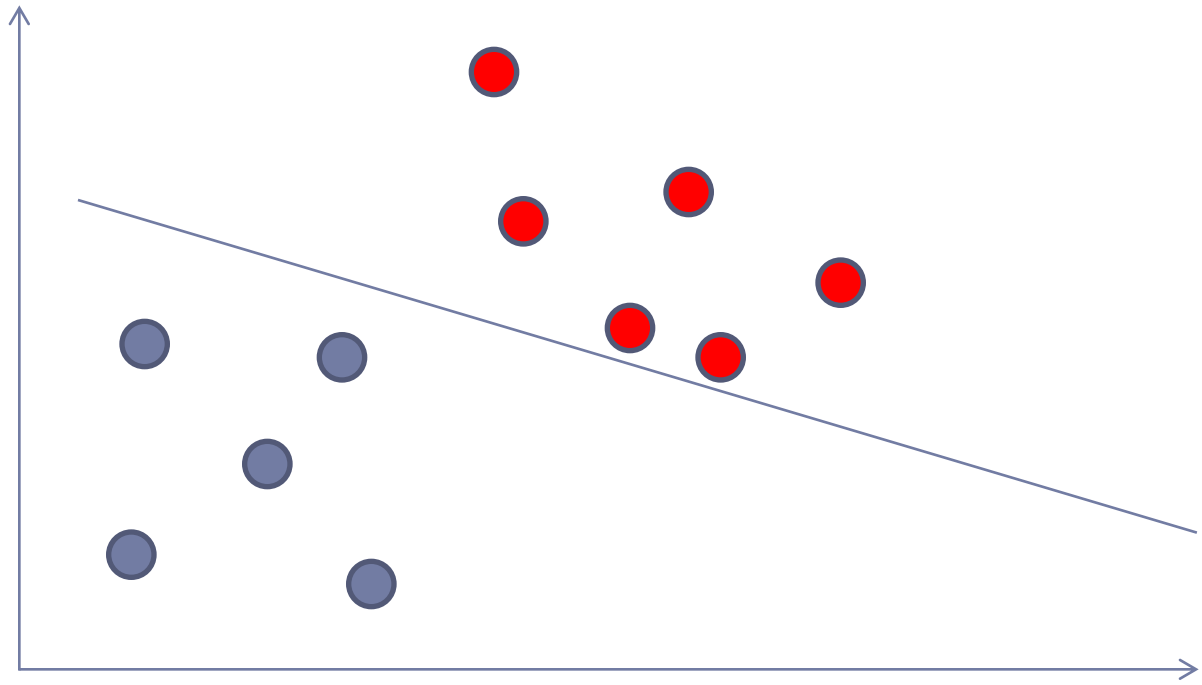
Support Vector Machines

- Máquinas de Vetores Suporte
- Classificação e Regressão
- Busca encontrar a melhor superfície de separação entre os dados de diferentes classes

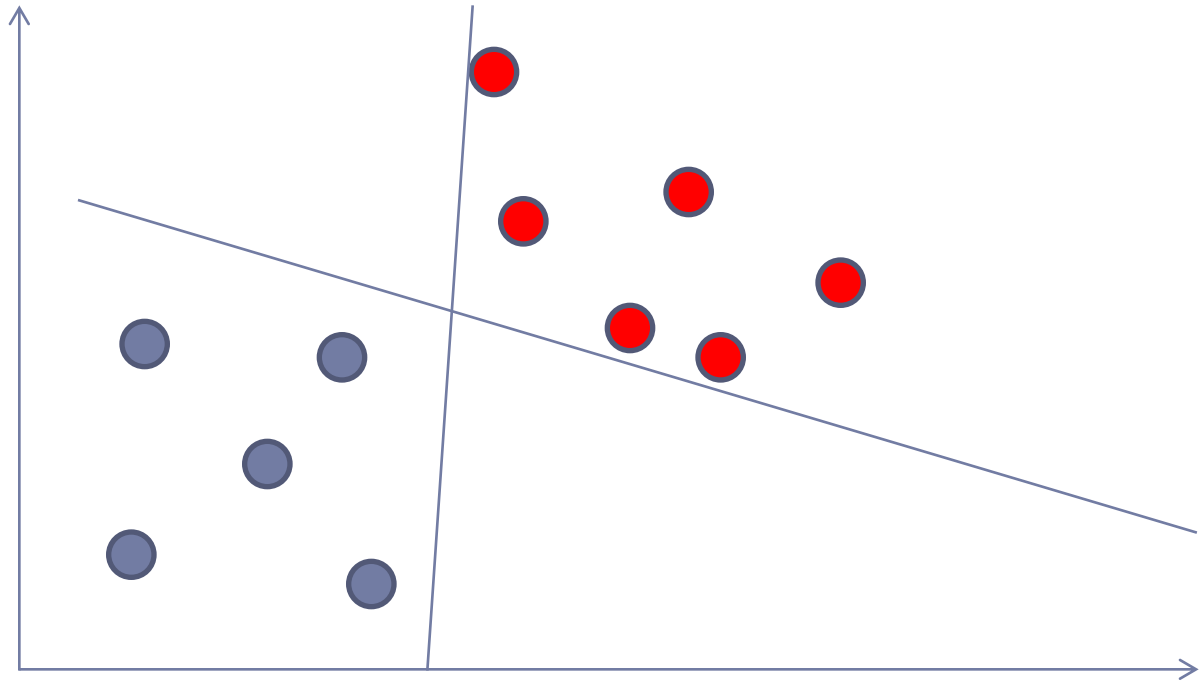
Superfície de Separação



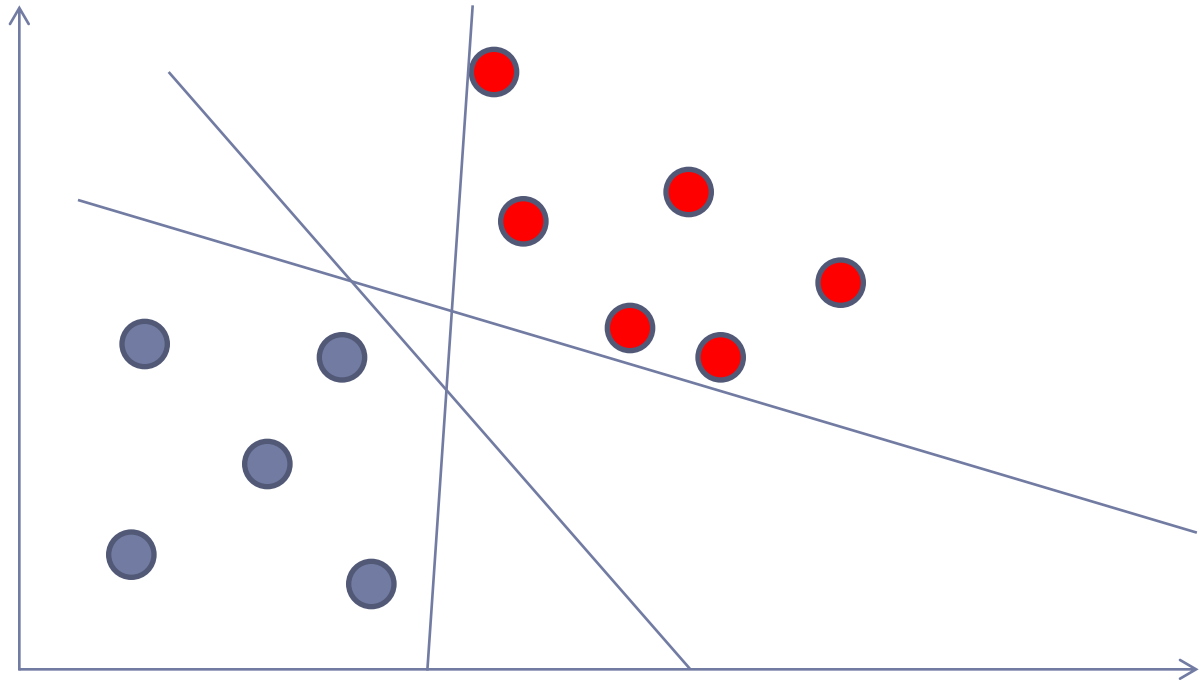
Superfície de Separação



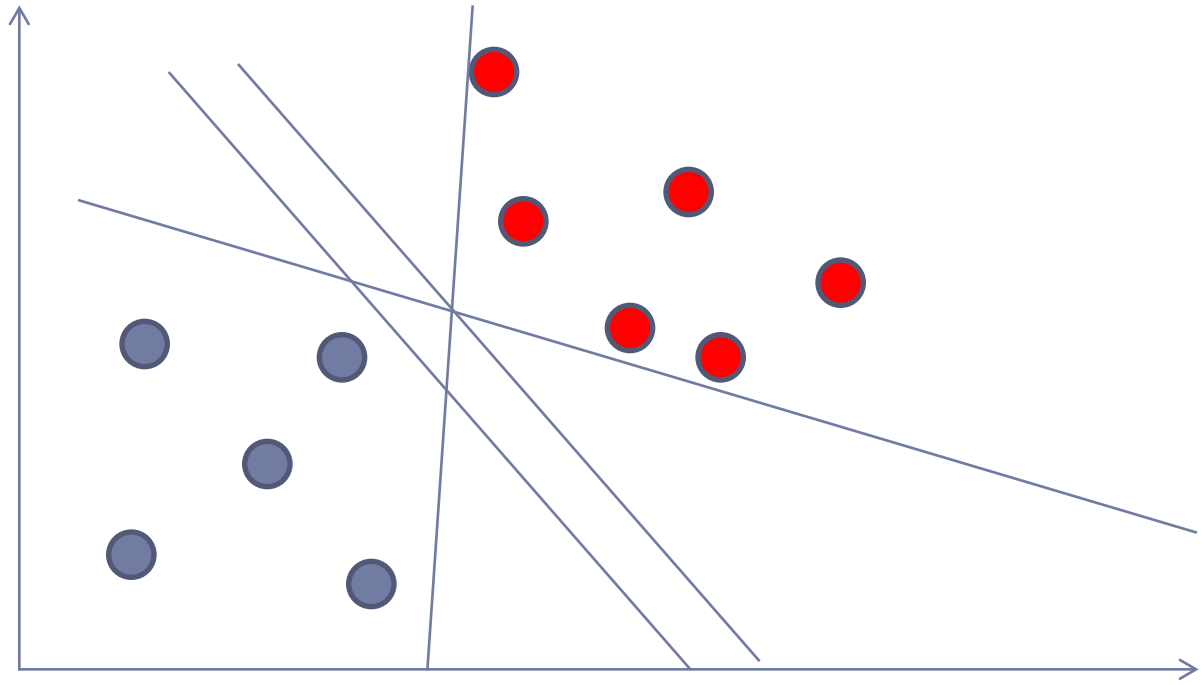
Superfície de Separação



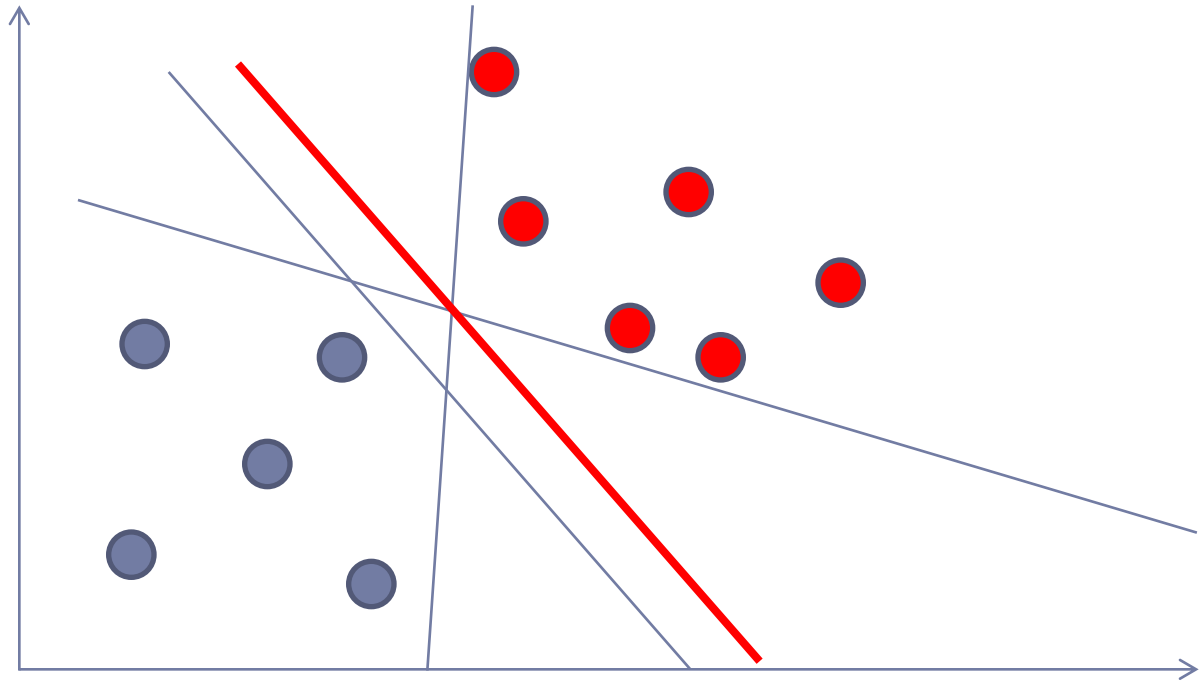
Superfície de Separação



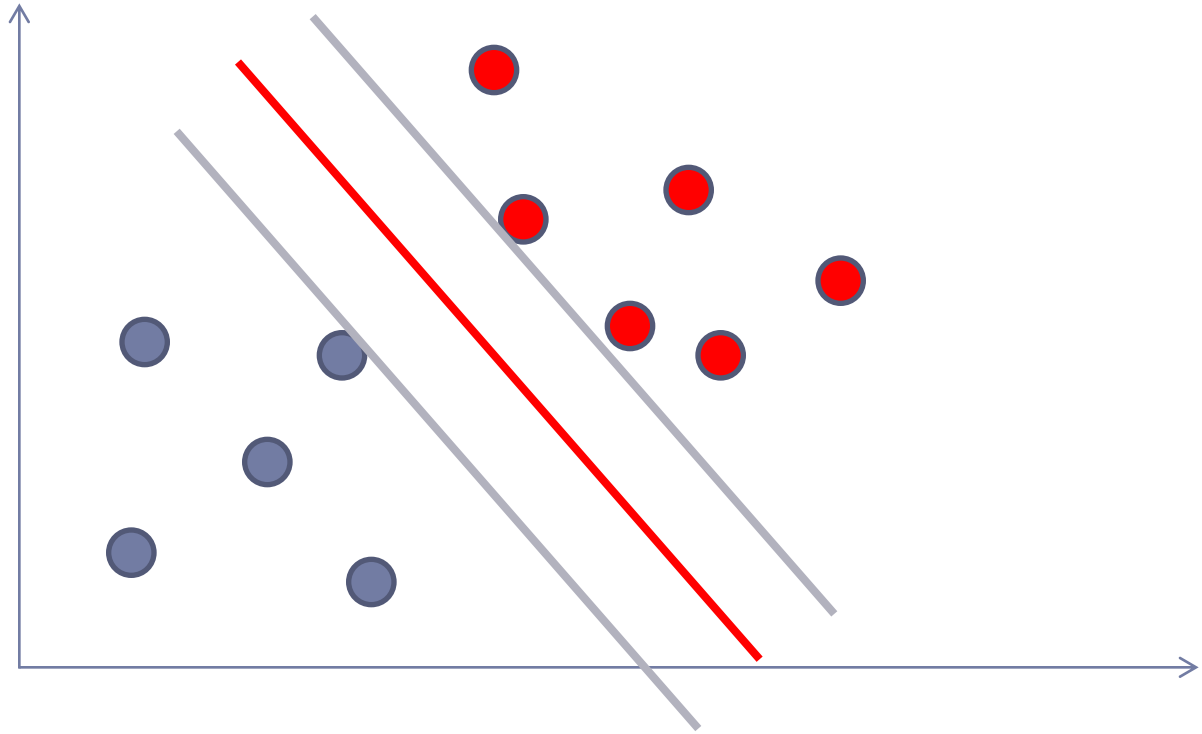
Superfície de Separação



Superfície de Separação



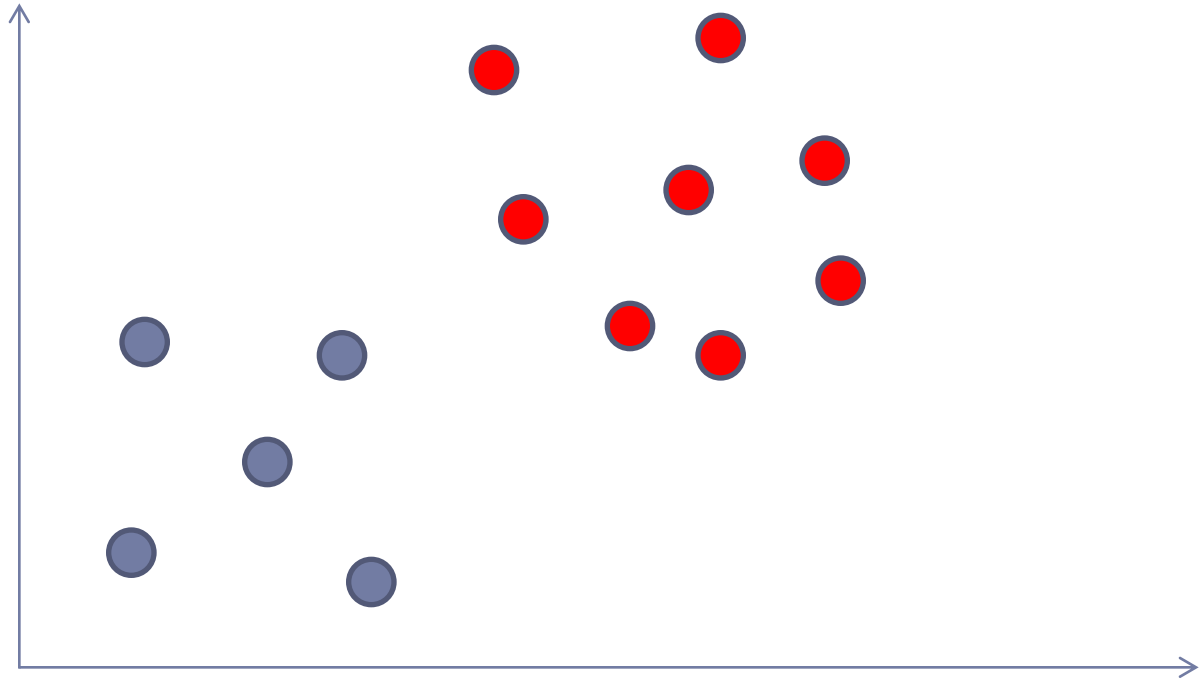
Margem de Separação



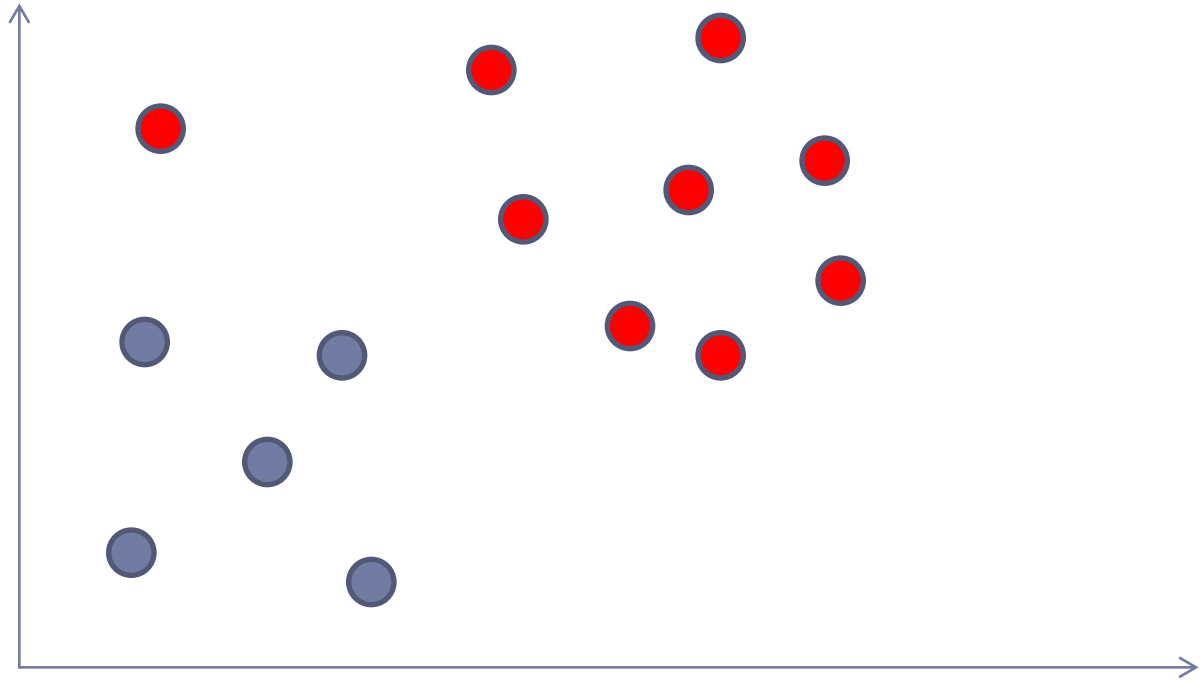
Support Vector Machines

- Maximizar a margem de separação dos dados de treinemanto.
- Hyperparâmetros
  - Linear – C
  - Não Linear – Parâmetros do Kernel

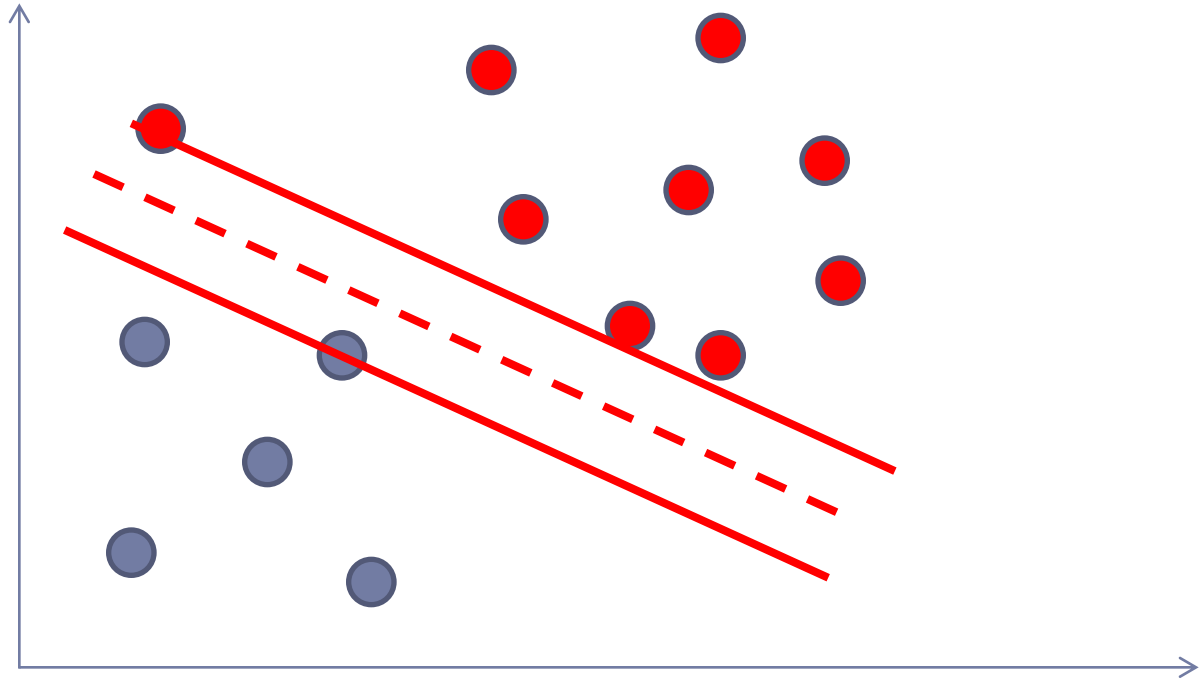
SVM – Parâmetro C



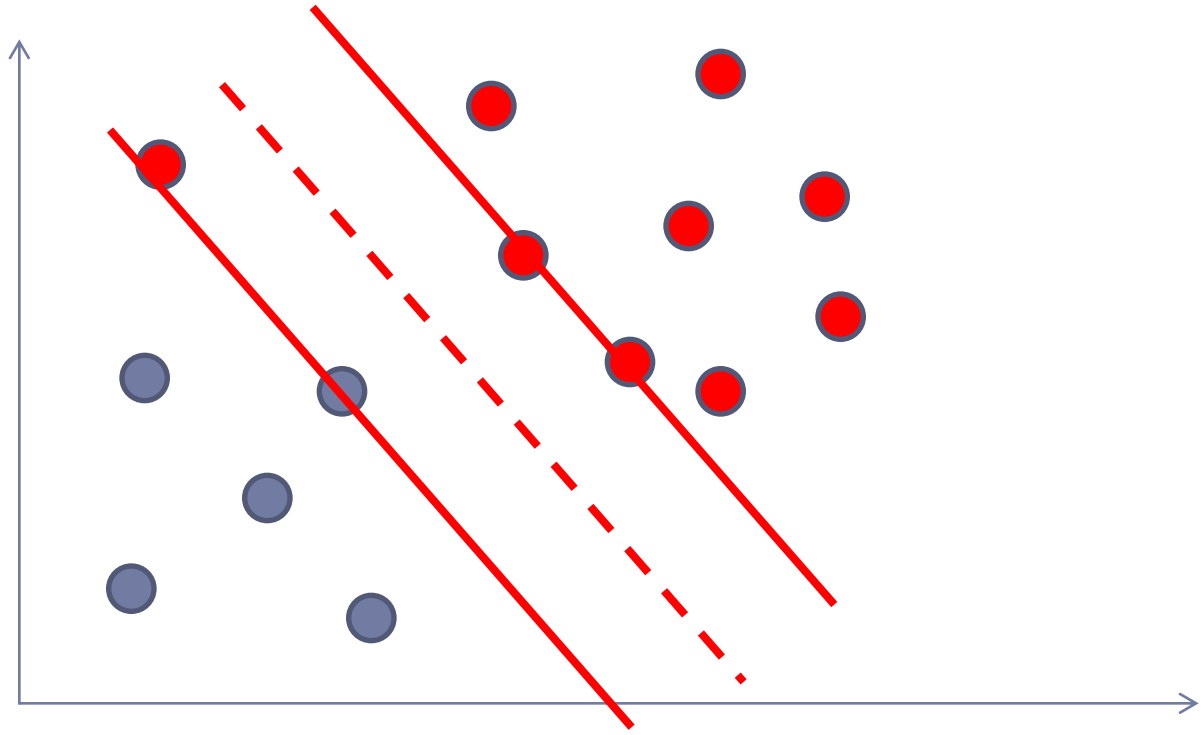
SVM – Parâmetro C



SVM – Parâmetro C



SVM – Parâmetro C



SVM – Parâmetro C

- Define o custo do seu modelo errar alguns exemplos de treinamento
- Errar pode ser bom desde que a margem seja aumentada
- No scikit learn
  - Inverso da logica do custo
  - Menor C -> Muito ajustado ao treinamento -> Poucos erros no treino

```
from sklearn.svm import SVC
clf = SVC(kernel = 'linear', C=this_C).fit(X_train, y_train)
```

Modelos Lineares

- Vantagens
  - Treinamento simples
  - Interpretáveis
  - Rápidos
  - Funcionam bem para grandes massas de dados
- Desvantagens
  - Para dados de baixa dimensão, modelos não-lineares podem ter melhor performance
  - Dados podem não ser linearmente separáveis

Exercício

- Carregar os dados do dataset breast cancer
- Classificar usando todos os modelos lineares vistos até agora

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)
```