

Método dos K vizinhos mais próximos

Fernando Henrique S. Barreto

IMECC - UNICAMP
Programa Estágio Docente (PED)

16 de novembro de 2022

- 1 Introdução
- 2 Algoritmo K-NN
- 3 Referências

Modelos generativos vs discriminativos

- Devroye et al. (1996) [1]:
 - Jargões preferidos;
 - medida μ , probabilidade a posteriori $\eta(x) = \mathbb{E}(\mathbf{Y}|\mathbf{X} = x)$;
 - um **classificador** $g(\mathbf{X}) : \mathbb{R}^d \rightarrow \{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$;
 - × um **erro de classificação** $g(\mathbf{X}) \neq \mathcal{C}_k$; e
 - ✓ o **classificador ótimo** $g^*(\mathbf{X}) \leq g(\mathbf{X})$.

Classificador de Bayes

$$g^*(x) = \begin{cases} 1, & \eta(x) > \frac{1}{2} \\ 0, & \text{caso contrário} \end{cases} .$$

A função g^* minimiza o risco $L^* = \mathbb{P}(g(\mathbf{X}) \neq \mathbf{Y})$.

Modelos Gaussianos

- LDA: $\mathbf{X} = \mathbf{X} | C_k \sim N_p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$; e
- QDA: $\mathbf{X} = \mathbf{X} | C_k \sim N_p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

Decision boundary

- LDA (linear):

$$\delta_k(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_k + \log(\pi_k),$$

- QDA (quadrático):

$$\delta_k(\mathbf{x}) = -\frac{1}{2} \mathbf{x}^\top \boldsymbol{\Sigma}_k^{-1} \mathbf{x} + \mathbf{x}^\top \boldsymbol{\Sigma}_k^{-1} \boldsymbol{\mu}_k - \frac{1}{2} \boldsymbol{\mu}_k^\top \boldsymbol{\Sigma}_k^{-1} \boldsymbol{\mu}_k + \log(\pi_k).$$

Bishop (2006):

- link: $a_k = a_k(\mathbf{x}) = \mathbf{w}_k \mathbf{x}^\top + \omega_{k0}$; e
- sigmoid:

$$\sigma(a_k) = \frac{e^{a_k}}{\sum_j e^{a_j}} = \frac{\exp(\mathbf{w}_k \mathbf{x}^\top + \omega_{k0})}{\sum_j \exp(\mathbf{w}_j \mathbf{x}^\top + \omega_{j0})} \implies \sigma(a_k) \equiv P(C_k | \mathbf{x}).$$

Murphy (2012) - Estimadores de máxima verossimilhança

- $\hat{\pi}_k = \frac{N_k}{N}$;
- $\hat{\boldsymbol{\mu}}_k = \frac{1}{N_k} \sum_{i: y_i=k} \mathbf{x}_i$; e
- $\hat{\boldsymbol{\Sigma}}_k = \frac{1}{N_k} \sum_{i: y_i=k} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k) (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)^\top$.

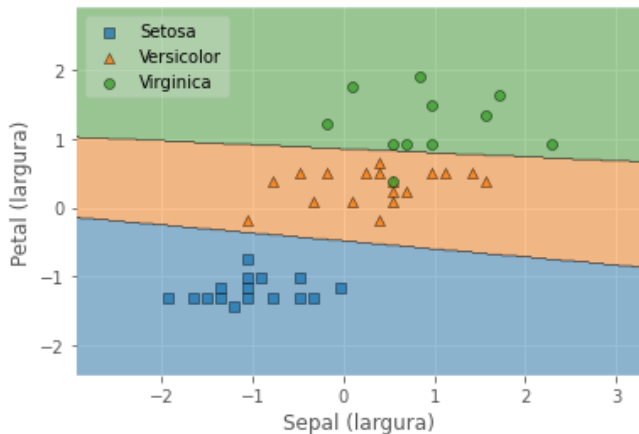


Figura: LDA: Sepal vs Petal (larg.)

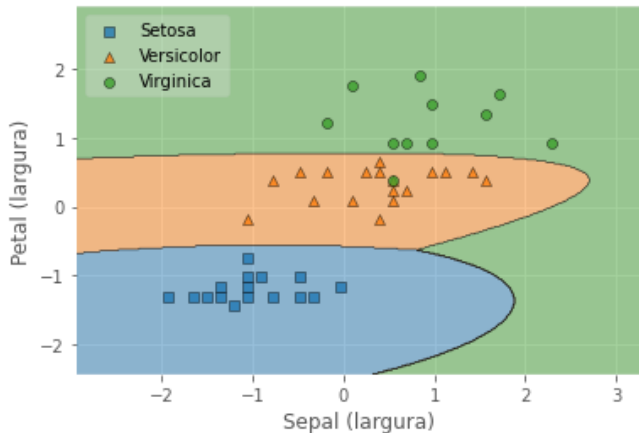


Figura: QDA: Sepal vs Petal (larg.)

Motivação

Esse exemplo foi construído baseado em um estudo de simulação apresentado em [James et al. \(2013\)](#) [2].

Geramos 500 bancos de dados formados por 5 classes e inputs trivariados sob três cenários, ou seja, a dimensão dos dados é 250×4 . Os cenários são distinguidos por

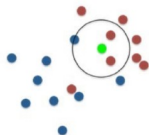
- (a) cenário I: os *inputs* $\mathbf{x} \sim N_3(\boldsymbol{\mu}_k, \mathbf{I})$ com classes são razoavelmente bem separadas;
- (b) cenário II: os *inputs* $\mathbf{x} \sim N_3(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ com classes são fortemente bem separadas; e
- (c) cenário III: os *inputs* \mathbf{x} seguem distribuição $t(\text{g.l.} = 2)$ com classes são fracamente bem separadas.

Tabela: Comparativo de performance entre os modelos

Método	Taxa de erro (média)		
	Cenário I	Cenário II	Cenário III
LDA	0,095	0,043	0,368
Naive Bayes	0,096	0,049	0,358
QDA	0,096	0,039	0,376

- Simples, mas um bom algoritmo supervisionado!
- Classificador baseado em medida de similaridade (Classificadores LDA e QDA também podem ser baseados em similaridade).
- O método K-NN funciona bem quando classes semelhantes estão agrupadas em torno de certos sub-espços dos inputs.
- No entanto, uma primeira grande desvantagem no método é seu possível custo computacional.

- Uma dedução ingênua para o classificador é prever a classe de uma observação \mathbf{x}_0 pela classe que é mais frequentemente representada entre os K vizinhos mais próximos.





- Em outras palavras, estimando as probabilidades posteriores $\mathbb{P}(C_i|\mathbf{x})$ pelo “VOTO MAJORITÁRIO” das classes C_i entre os K vizinhos.
- Para simplificar, daqui em diante, admitamos apenas duas classes. Para evitar problemas de empate nos votos, tomemos apenas valores ímpares para K .

Algoritmo K-NN

A

y: 

Voto Majoritário: 
Voto Popular: 

B

y: 


Voto Majoritário: Ninguém
Voto Popular: 

Figura: Exemplo de votação

- Se $K = 1$, o classificador atribui ao ponto de teste a classe do vizinho mais próximo.
- O classificador 1-NN pode ser surpreendentemente útil se o tamanho da amostra for suficientemente grande.
- Entretanto, o classificador 1-NN pode ter um fraco desempenho em casos de amostras pequenas devido ao *overfitting*.
- Ele também tende a ter um desempenho ruim caso multi-classes.
- Sob certas condições, há evidências de que os classificadores 3-NN e 5-NN performam melhor que 1-NN.

Notações e pseudocódigo

Fixando $\mathbf{X}_0 \in R^P$, reordene $(Y_1, \mathbf{X}_1), (Y_2, \mathbf{X}_2), \dots, (Y_N, \mathbf{X}_N)$ de acordo com os valores crescentes de $\|\mathbf{X}_i - \mathbf{X}_0\|_2$. A notação do conjunto de dados reordenado é

$$(Y_{(1)}, \mathbf{X}_{(1)}), (Y_{(2)}, \mathbf{X}_{(2)}), \dots, (Y_{(N)}, \mathbf{X}_{(N)}).$$

Formalmente, o classificador $K - NN$ é definido por

$$g_n(\mathbf{X}_0) = \begin{cases} 1, & \sum_{i=1}^n w_{ni} I\{Y_i = 1\} > \sum_{i=1}^n w_{ni} I\{Y_i = 0\} \\ 0, & \text{caso contrário} \end{cases},$$

com $w_{ni} = 1/k$ se \mathbf{X}_i é um dos K vizinhos de \mathbf{X}_0 ou $w_{ni} = 0$, caso contrário.

Algoritmo K-NN

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Wed Nov 16 07:48:03 2022
4
5 @author: Fernando
6 """
7
8 from sklearn.model_selection import train_test_split
9 from sklearn.metrics import confusion_matrix
10 from sklearn.datasets import make_classification
11 import matplotlib.pyplot as plt
12 import numpy as np
```

Algoritmo K-NN

```
1 X, Y = make_classification(n_samples = 100,  
2                           n_features = 2,  
3                           n_informative =2,  
4                           n_redundant =0,  
5                           n_repeated =0,  
6                           n_classes = 3,  
7                           n_clusters_per_class=1,  
8                           weights = [.3,.3,.4],  
9                           class_sep=1.8,  
10                          random_state = 3)  
11  
12  
13 X_train, X_test, y_train, y_test = \  
14     train_test_split(X, Y, test_size=0.33, random_state  
    =42)
```


Algoritmo K-NN

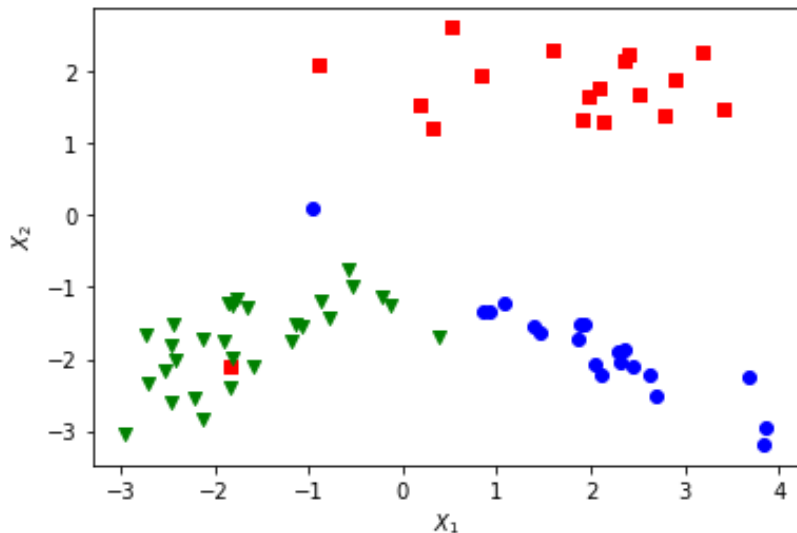


Figura: Conjunto de treinamento

Pseudocódigo do algoritmo K-NN

Input: D , the set of k training objects, and test object $z = (\mathbf{x}', y')$

Process:

Compute $d(\mathbf{x}', \mathbf{x})$, the distance between z and every object, $(\mathbf{x}, y) \in D$.

Select $D_z \subseteq D$, the set of k closest training objects to z .

Output: $y' = \underset{v}{\operatorname{argmax}} \sum_{(\mathbf{x}_i, y_i) \in D_z} I(v = y_i)$

- O pseudocódigo acima é o mais simples possível para o algoritmo K-NN.
- Observe que o pseudo código acima não pondera pelo peso $w_{ni} = 1/k$. Detalhe: se K puder variar com N desde que $K_N/N \rightarrow 0$, então pesos w_{ni} desbalanceados até podem ser vantajosos.

Distâncias

Distância de Minkowski:

$$\|\mathbf{X}\|_p = \sum_{i=1}^n (|x_1|^p + |x_2|^p + \dots + |x_n|^p)^{1/p}$$

- se $p = 2$, então norma euclidiana.
- se $p = 1$, então norma de Manhattan.

Distância de Hamming: é o número de posições em que os símbolos correspondentes são diferentes entre duas strings de igual comprimento. Exemplo: $\mathbf{a} = (0, 1, 1, 1, 1)^\top$ e $\mathbf{b} = (0, 1, 0, 1, 1)^\top$ implica que $d(\mathbf{a}, \mathbf{b}) = 1$.

Cosine similarity: $\cos(\theta) = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$

Algoritmo K-NN

```
1 def euclidean(point, data):
2     return np.sqrt(np.sum((point - data)**2))
3
4 K=3 # 3 Vizinhos mais proximos
5 pred=np.array(len(y_test)*[0])
6
7 for i in range(len(y_test)):
8     d = np.zeros(X_train.shape[0])
9     for j in range(X_train.shape[0]):
10        d[j] = euclidean(X_test[i,], X_train[j,])
11        kNN=np.sort(d)[0:K]
12        loc = np.zeros(len(kNN))
13        for u in range(len(kNN)):
14            loc[u] = np.where(kNN[u]==d)[0][0]
15        unique, counts = \
16            np.unique(y_train[loc.astype(int)],
17                      return_counts=True)
18        pred[i] = unique[np.where(np.max(counts)==counts)
19                          [0][0]]
```

Algoritmo K-NN

IPython Console



Console 1/A x

```
In [10]: # Matriz confusao
```

```
In [11]: confusion_matrix(y_test, pred)
```

```
Out[11]:  
array([[13,  0,  1],  
       [ 0,  8,  2],  
       [ 0,  0,  9]], dtype=int64)
```

```
In [12]: St=np.sum(confusion_matrix(y_test, pred))
```

```
In [13]: Sd=np.sum(np.diag(confusion_matrix(y_test, pred)))
```

```
In [14]: ER=np.round((St-Sd)/St * 100,2);ER #Taxa de erro
```

```
Out[14]: 9.09
```

```
In [15]: |
```

Algoritmo K-NN

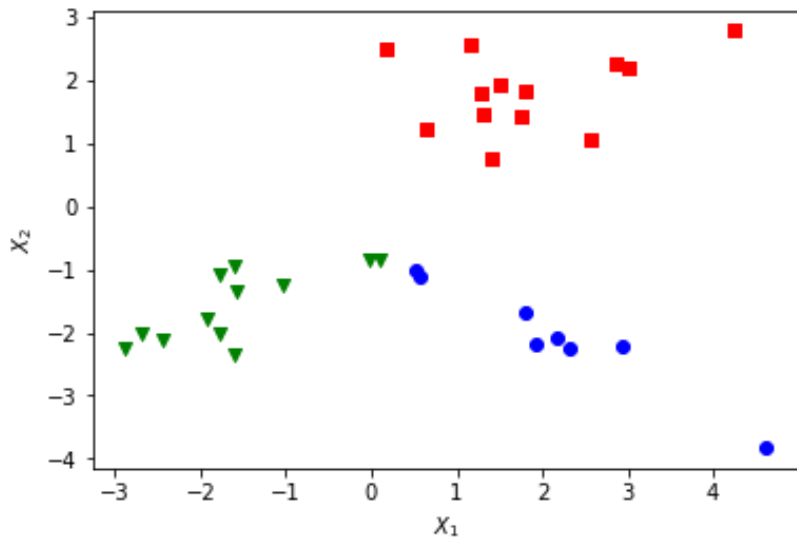


Figura: Conjunto de teste

Tabela: Comparativo de performance entre os modelos

Método	Média das taxas de erro (erro padrão)		
	Cenário I	Cenário II	Cenário III
LDA	0,095 (0,10)	0,043 (0,09)	0,368 (0,12)
Naive Bayes	0,096 (0,10)	0,049 (0,08)	0,358 (0,11)
QDA	0,096 (0,11)	0,039 (0,08)	0,376 (0,12)
5-NN	0,095 (0,10)	0,040 (0,10)	0,126 (0,10)

Consistência

- O algoritmo é consistente sob certas condições. Para requerer consistência, é preciso verificar se $\lim_{N \rightarrow \infty} K_N/N = 0$ ou \mathbf{X} seja independente do conjunto de treino sempre que.

Lema 5.1. Devroye (1996) pág. 63

Se \mathbf{X}_0 pertence ao suporte da distribuição de \mathbf{X} e $\lim_{N \rightarrow \infty} K_N/N = 0$, então $\|\mathbf{X}_{(K)} - \mathbf{X}_0\| \rightarrow 0$ com probabilidade 1. Se \mathbf{X} é iid. do conjunto de treino, então $\|\mathbf{X}_{(K)} - \mathbf{X}_0\|$ converge para 0 com probabilidade 1 sempre que $\lim_{N \rightarrow \infty} K_N/N = 0$.

Prova intuitiva em sala de aula.

Algoritmo K-NN

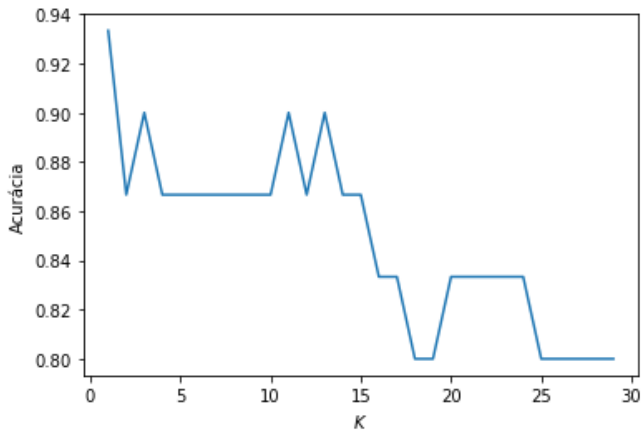


Figura: Inconsistência do K-NN

Algoritmo K-NN

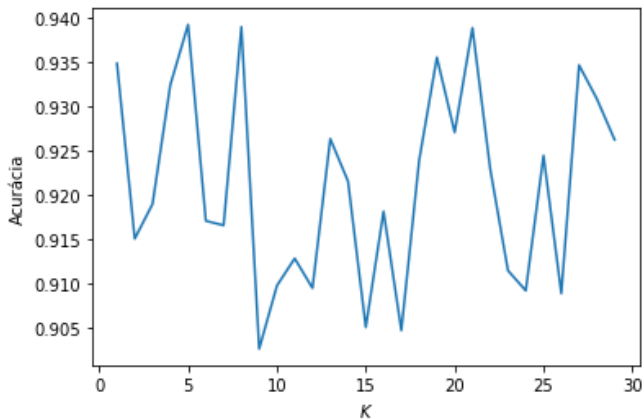


Figura: $\lim_{N \rightarrow \infty} K_N/N = 0$

Seleção de modelo

Como devem ter notado, existem maneiras diferentes de como podemos melhorar o desempenho preditivo do algoritmo K-NN:

- Escolhendo o valor de K .
- Padronizando as escalas dos inputs.
- Escolher a medida de distância mais apropriada.
- Ponderação da medida de distância.

Algoritmo K-NN

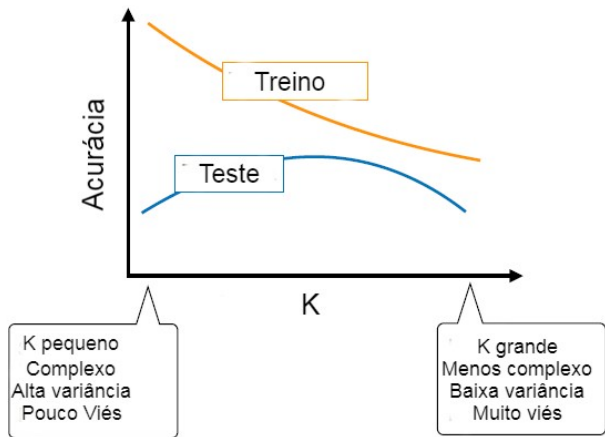


Figura: Trade-off com relação a complexidade do modelo

Algoritmo K-NN

```
1 from sklearn.datasets import load_iris
2 from sklearn.model_selection import train_test_split
3 from sklearn.neighbors import KNeighborsClassifier
4 iris = load_iris()
5 X = iris.data
6 y = iris.target
7 # Dividir o conjunto de dados em treino e teste
8 X_train, X_test, y_train, y_test = \
9     train_test_split(X, y, random_state=4)
10 knn = KNeighborsClassifier(n_neighbors = 5)
11 knn.fit(X_train, y_train)
12 y_pred = knn.predict(X_test)
13
14 confusion_matrix(y_test, y_pred)
15 St=np.sum(confusion_matrix(y_test, y_pred))
16 Sd=np.sum(np.diag(confusion_matrix(y_test, y_pred)))
17 ER=np.round((St-Sd)/St * 100,2);ER #Taxa de erro
18
19 print('Erro: ', np.round(1-knn.score(X_test, y_test)
20     ,4)*100)
20 # Erro:  2.63
```

Algoritmo K-NN

```
1 from sklearn.model_selection import cross_val_score
2 knn = KNeighborsClassifier(n_neighbors = 5)
3 scores = cross_val_score(knn, X, y, cv=5, scoring='
    accuracy')
4 print(1-scores)
5 # [0.03333333 0.  0.06666667 0.03333333 0. ]
6 print(1-scores.mean())
7 # 0.02666666666666
```

```
1 k_range = range(1, 31)
2 k_scores = []
3 for k in k_range:
4     knn = KNeighborsClassifier(n_neighbors=k)
5     scores = cross_val_score(knn, X, y, cv=5, scoring='
    accuracy')
6     k_scores.append(scores.mean())
```

Algoritmo K-NN

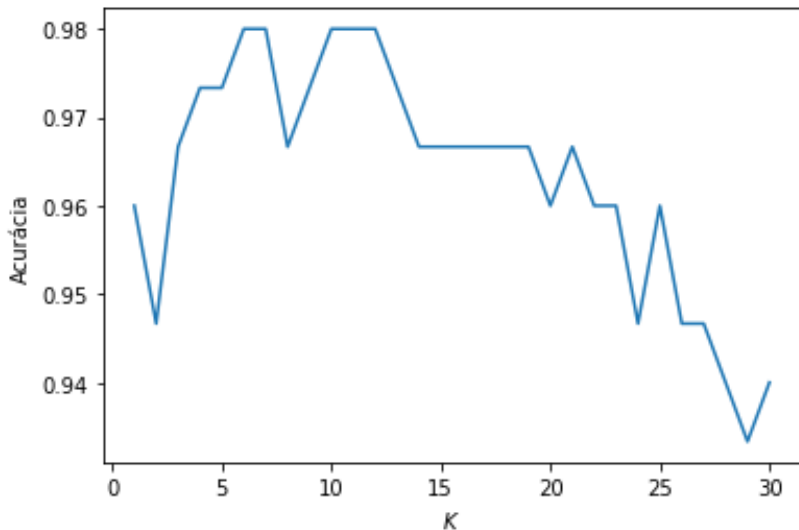


Figura: Seleccionando K via VC

- Ao dividir o conjunto de dados em treino e teste, temos garantido que estes seguem iid.
- Queremos que o classificador seja correlacionado e independente a $\mathbf{Y}_{\text{teste}}$ condicionado a $\mathbf{X}_{\text{teste}}$.
- Pelo método CV, selecionamos o K que minimiza a taxa de error de classificação.
- Nem sempre utilizar o método CV será útil.



Desvantagens

Apesar de ser relativamente fácil de implementar e interpretar, o algoritmo pode ter

- problemas com dados em alta dimensão;
- um custo computacional considerável; e
- nem sempre é uma tarefa trivial escolher a distância mais apropriada ao conjunto de dados.

Entretanto,

- uso de estrutura de dados como KD-trees e Ball-trees podem tornar o K-NN substancialmente mais eficiente; e
- comparado a outros algoritmos de aprendizado, o método K-NN possui 2 hiperparâmetros.

-  Luc Devroye, László Györfi, and Gábor Lugosi.
A probabilistic theory of pattern recognition, volume 31.
Springer Science & Business Media, 1996.
-  Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani.
An introduction to statistical learning, volume 112.
Springer, 2013.

Obrigado!