Exercício 7

Author: Flavio Altinier

RA 135749

In [1]:
```
import numpy as np
import plotly.offline as py
import plotly.graph_objs as go

inputFilesN = range(1, 6)
inputFileName = 'input/serie'

py.init_notebook_mode(connected=True)
```

Lendo Dados

In [2]:
```
dataSeries = []
for i in inputFilesN:
    data = np.loadtxt(fname = inputFileName + str(i) + '.csv',\
                      delimiter=',',\
                      skiprows=1,\
                      usecols=(1,))
    dataSeries.append(data)
```

Extraindo Samples

A ideia é não fixar o tamanho da janela de sampling, mas decidi-la conforme os dados vizinhos -- se estiverem dentro de um threshold, estarão dentro do mesmo sample. O tamanho da janela será então, também, uma feature.

In [3]:
```
epsilon = 0.2 # threshold sera 20% do espaco vertical
sampledData = []

for data in dataSeries:
    localMax = np.max(data)
    localMin = np.min(data)
    threshold = abs(localMax-localMin) * epsilon
    X = []
    currentBatch = []
    batchMean = data[0]
    for i in range(data.shape[0] - 1):
        if abs(batchMean - data[i]) < threshold: # ponto pertence a batch atual
            currentBatch.append(data[i])
            batchMean = np.mean(np.asarray(currentBatch))
        else: # salva batch atual e cria um novo
            X.append([batchMean, len(currentBatch)])
            batchMean = data[i]
            currentBatch = [data[i]]
    sampledData.append(X)
```

Agora temos, em sampledData, para cada série, os conjuntos de pontos vizinhos com valores proximos, no formato (mediaDosPontos, numeroDePontos).

Funções de Visualização

Uma vez extraídos os samples de cada grupo de dados, definimos as funcoes que vao nos auxiliar na visualização:

In [4]:
```
def visualizeResultsBinary(sampledData, dataSeries, index, labels, title):
    colors = np.zeros(dataSeries[index].shape[0])
    count = 0
    traces = []
    layout = go.Layout(title=title)
    for i in range(len(sampledData[index])):
        colors[count:count+sampledData[index][i][1]] = labels[i]
        count = count + sampledData[index][i][1]
    itemCount = np.bincount(colors.astype(int))
    anomalyCluster = np.argmin(itemCount)
    normalPointIndices = np.where(colors != anomalyCluster)[0]
    trace1 = go.Scatter(x=normalPointIndices, y = dataSeries[index][normalPointIndices], mode='markers',\
                        marker=dict(color='rgb(0, 0, 0)'), \
                        name='Dados dentro do padrão')
    anomalyPointIndices = np.where(colors == anomalyCluster)[0]
    trace2 = go.Scatter(x=anomalyPointIndices, y = dataSeries[index][anomalyPointIndices], mode='markers',\
                       marker=dict(color='rgb(255, 25, 25)'), \
                       name = 'Dados anômalos')
    traces = [trace1, trace2]
    py.iplot(go.Figure(data=traces, layout=layout), filename='Clusters')

def visualizeResultsColored(sampledData, dataSeries, index, labels, title):
    colors = np.zeros(dataSeries[index].shape[0])
    count = 0
    traces = []
    for i in range(len(sampledData[index])):
        colors[count:count+sampledData[index][i][1]] = labels[i]
        count = count + sampledData[index][i][1]
    for i in range(np.max(labels) + 1): # para cada cluster
        localPointIndices = np.where(colors == i)[0]
        trace = go.Scatter(x=localPointIndices, y = dataSeries[index][localPointIndices], mode='markers')
        traces.append(trace)

    py.iplot(traces, filename='Clusters')
```

Cluster-Based Outlier Detection

A partir de agora, vamos explorar como um approach Cluster-Based é capaz de dividir os dados. Cada grupo de dados precisa de um pequeno tuning na detecção.

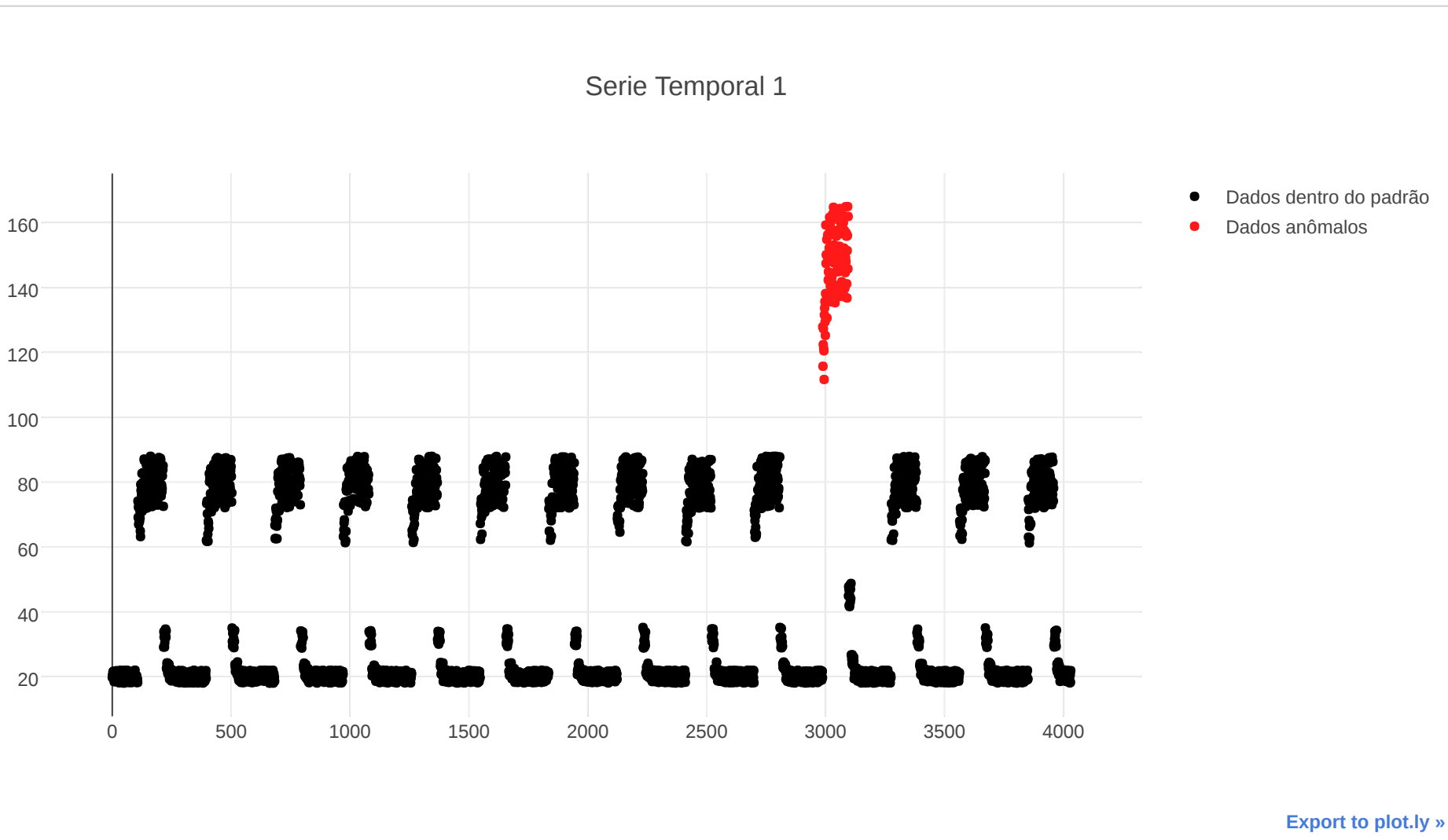
Serie Temporal 1

Nesta série um conjunto de dados se destaca como anômalo por ter valores maiores que o comum no eixo y. Para isso, um simples KMeans com 3 clusters é capaz de separar os dados anômalos:

In [5]:
```
from sklearn.cluster import KMeans

index = 0
curD = np.asarray(sampledData[index])

km = KMeans(n_clusters=3, init='random', random_state = 0)
labels = km.fit_predict(curD)
visualizeResultsBinary(sampledData, dataSeries, index, labels, 'Serie Temporal ' + str(index + 1))
```



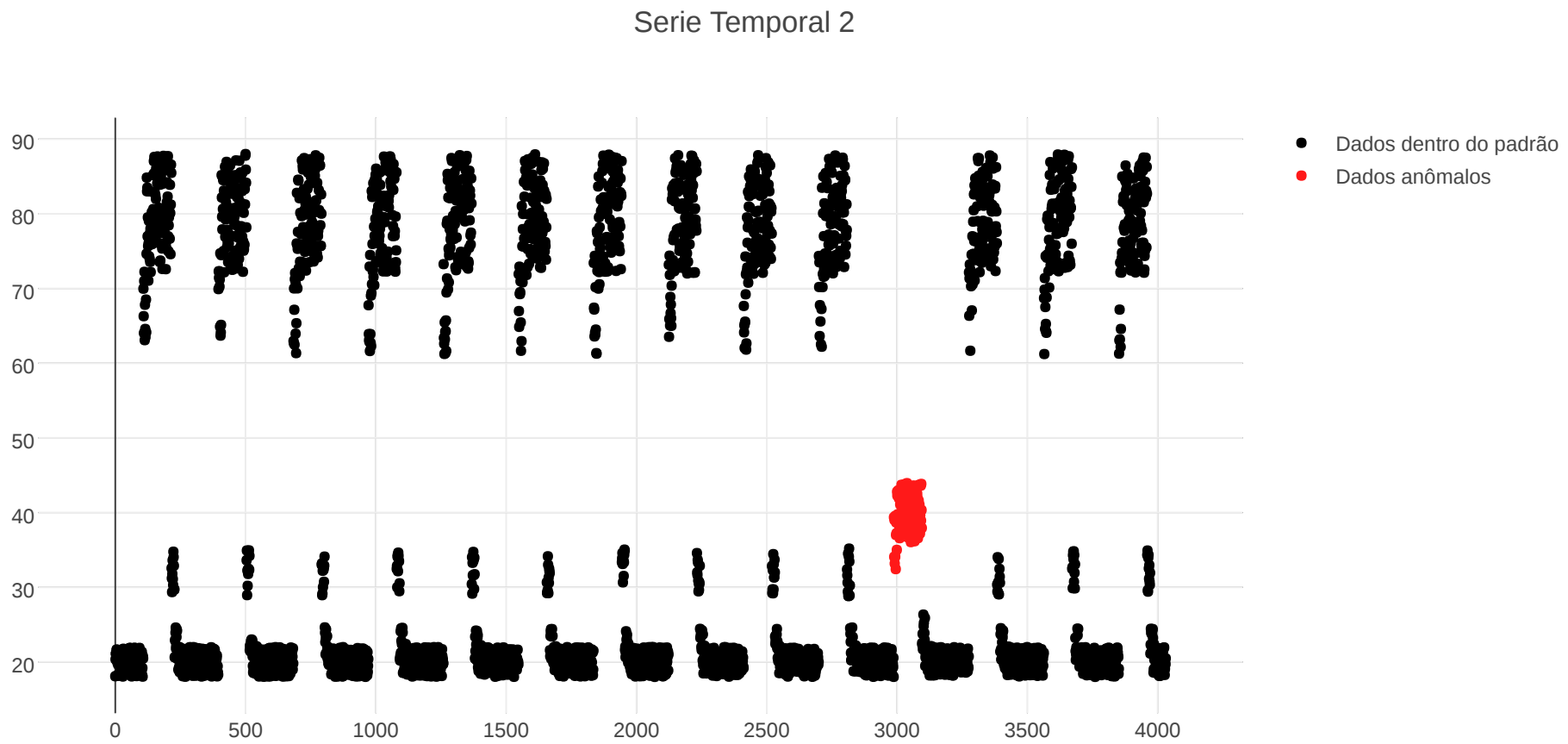
Observe como o KMeans foi capaz de facilmente encontrar o cluster anômalo nesta série.

Série Temporal 2

In [6]:
```
from sklearn.cluster import AffinityPropagation

index = 1
curD = np.asarray(sampledData[index])
curD = curD[:, 0].reshape(-1, 1)
ap = AffinityPropagation(damping = 0.51)
labels = ap.fit_predict(curD)

visualizeResultsBinary(sampledData, dataSeries, index, labels, 'Serie Temporal ' + str(index + 1))
```



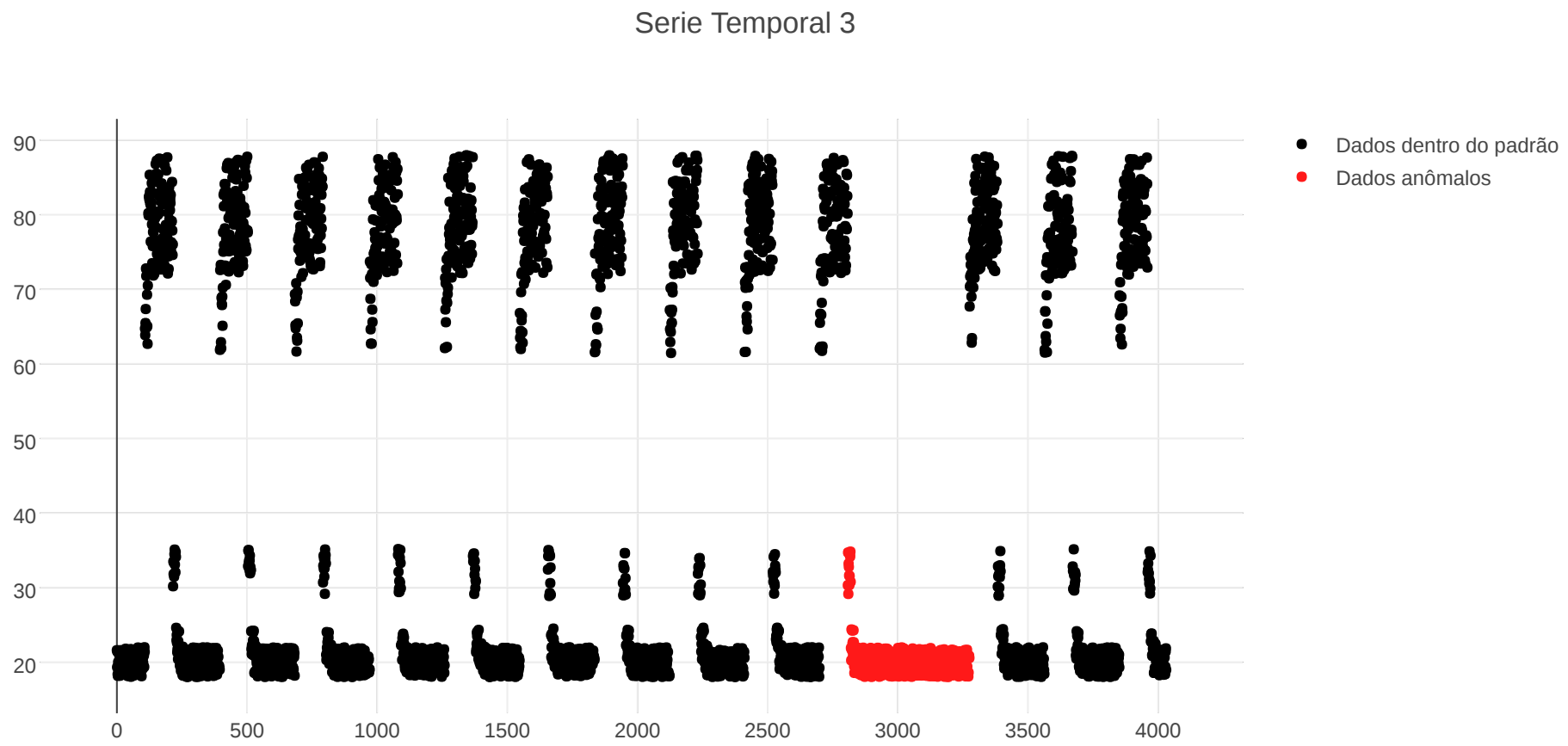
Export to plot.ly »

Neste caso foi utilizada a clusterização por Affinity Propagation -- um KMeans encontraria apenas os noise com valor por vlta de 65 como anômalos, pois o cluster anômalo está relativamente próximo da base. Affinity Propagation descobre esses noises como relacionado aos clusters superiores e é mais robusto neste caso, escolhendo corretamente o cluster anômalo.

Série Temporal 3

In [7]:
```
index = 2
curD = np.asarray(sampledData[index])

km = KMeans(n_clusters=3, init='random', random_state = 0)
labels = km.fit_predict(curD)
visualizeResultsBinary(sampledData, dataSeries, index, labels, 'Serie Temporal ' + str(index + 1))
```



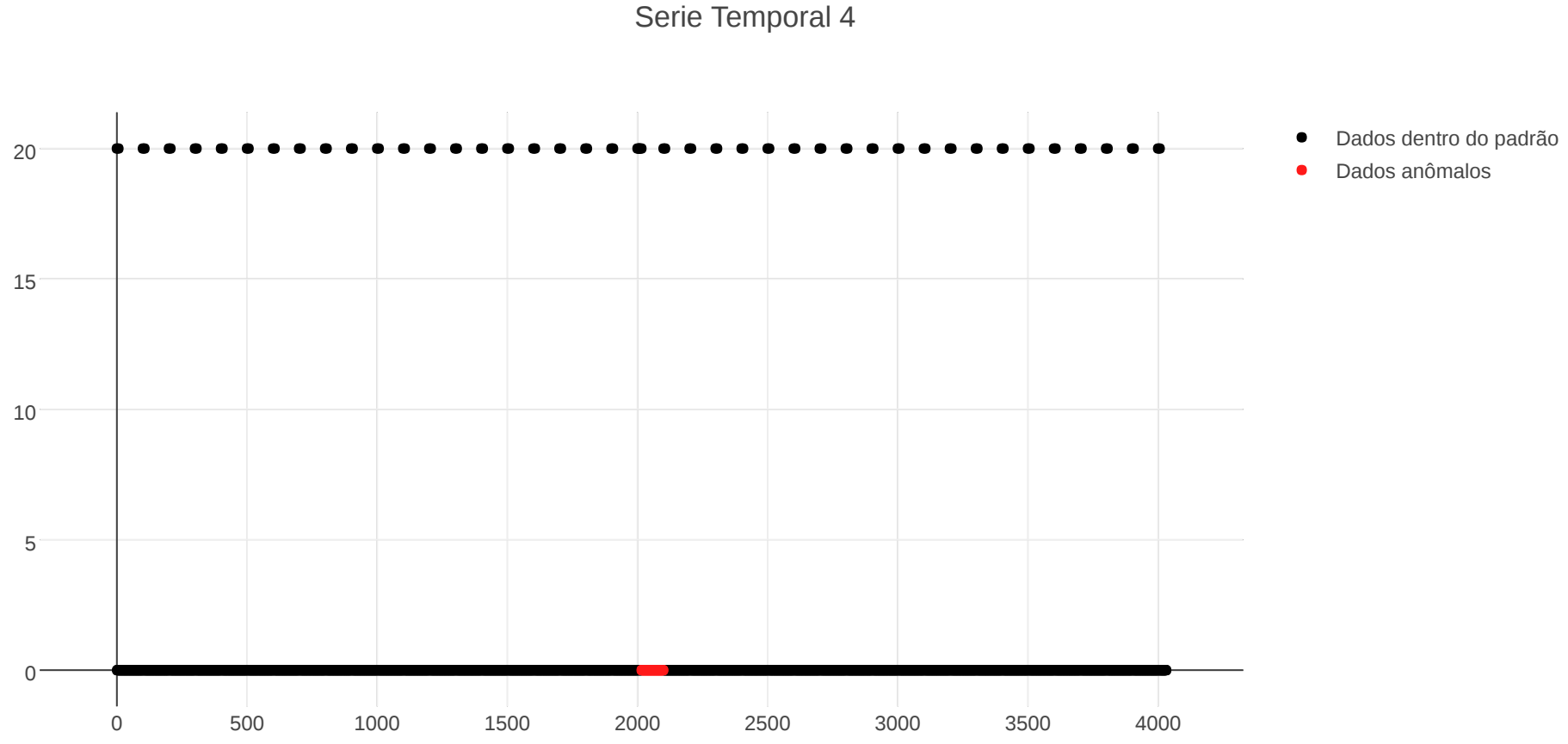
Export to plot.ly »

O KMeans simples com 3 clusters indica facilmente quais são os dados que fogem do padrão.

Série Temporal 4

In [8]:
```
index = 3
curD = np.asarray(sampledData[index])[:, 1].reshape(-1, 1)

km = KMeans(n_clusters=4, init='random', random_state = 0)
labels = km.fit_predict(curD)
visualizeResultsBinary(sampledData, dataSeries, index, labels, 'Serie Temporal ' + str(index + 1))
```



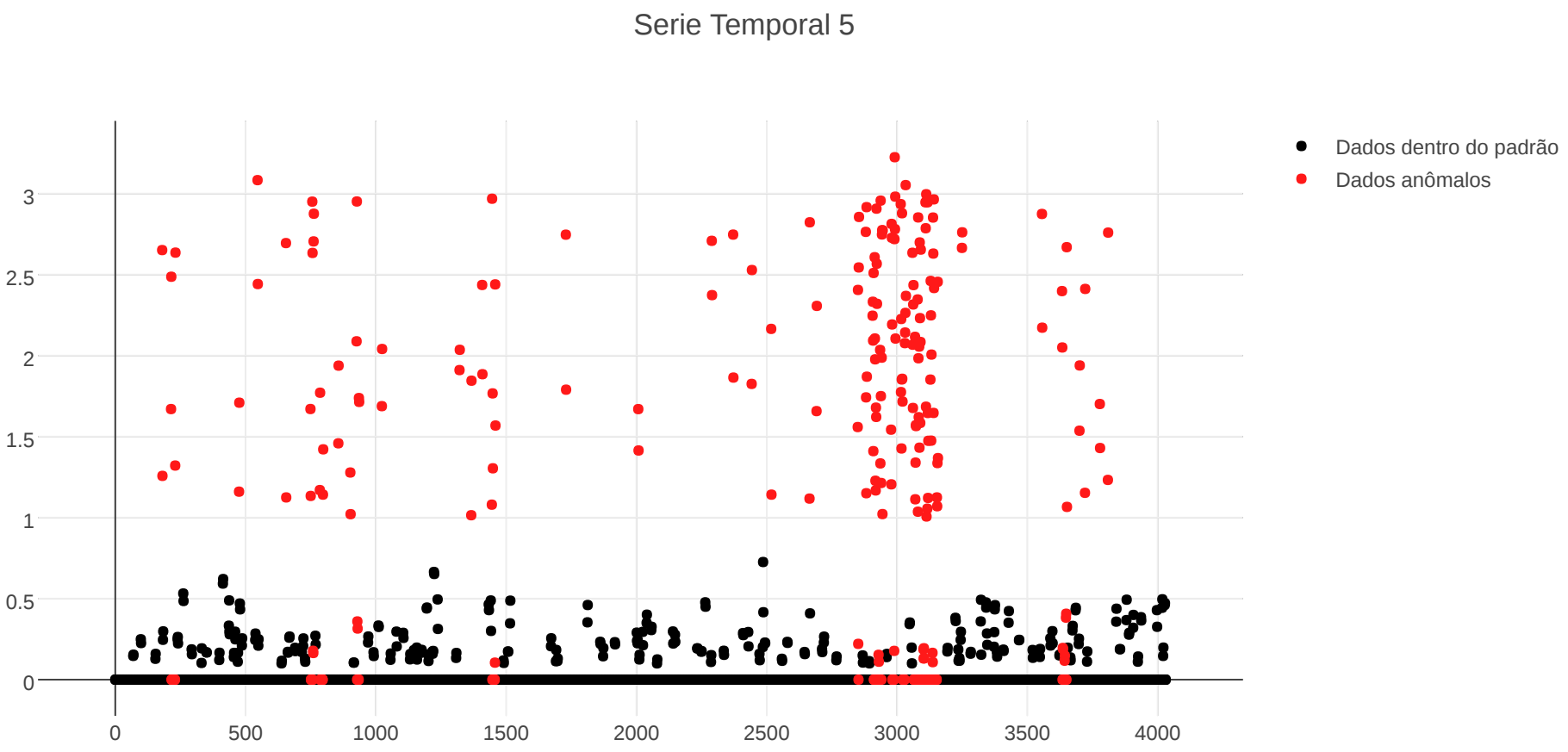
Export to plot.ly »

É de difícil visualização, mas há bastante noise na base: por isso precisamos aumentar em 1 o numer de clusters. Além disso, como não há anomalia no valor dos pontos em y mas sim no tamanho dos aglomerados, usamos apenas o tamanho do aglomerado como feature. Note que neste caso ele não acusou os picos duplicados como anomalia, mas a base desses picos. Provavelmente porque ali ficou um "buraco" maior, que fez com que esse intervalo na base fosse menor e ali foi detectada a anomalia.

Série Temporal 5: sem anomalias aparentes

In [9]:
```
index = 4
curD = np.asarray(sampledData[index])

km = KMeans(n_clusters=4, init='random', random_state = 0)
labels = km.fit_predict(curD)
visualizeResultsBinary(sampledData, dataSeries, index, labels, 'Serie Temporal ' + str(index + 1))
```



Export to plot.ly »

Observe como o algoritmo selecionou os dados anômalos: como a maioria dos dados varia entre 0 e 0.5, ele considerou esses dados como padrão. São poucos os maiores que 0.5 (em especial um cluster por volta dos dados 3000), e ele flaggeou esses dados como anômalos.