# Ex7

November 23, 2016

Exercicio 7 - Anomalias em séries temporais, MC886 - 2S-2016
Gabriel Hidasy Rezende, RA 116928

```
In [1]: # imports and constants
        import numpy as np
        from sklearn.cluster import KMeans
        from sklearn.ensemble import IsolationForest
        import matplotlib.pyplot as plt

        # Define window to correspond to one record
        # How many datapoints per section
        window = 50
        # Size of section intercept in each size
        intersection = 3
```

```
In [2]: def read_data():
            # Read data
            series = []
            for serie in [1, 2, 3, 4, 5]:
                with open("serie{}.csv".format(serie)) as f:
                    data = [float(x.split(",")[1])
                            for x in f.read().split("\n")[1:-1]]
                    series.append(data)

            # I will represent each records as (max, min, stddev, mean)
            t_series = []
            for serie in series:
                transformed_serie = []
                for i in range(0, len(serie), window - intersection):
                    w = serie[i:i + window]
                    maximum = max(w)
                    minimum = min(w)
                    mean = sum(w) / len(w)
                    stddev = np.std(np.array(w))
                    transformed_serie.append([maximum, minimum, mean])
                t_series.append(np.array(transformed_serie))
            return series, t_series
```

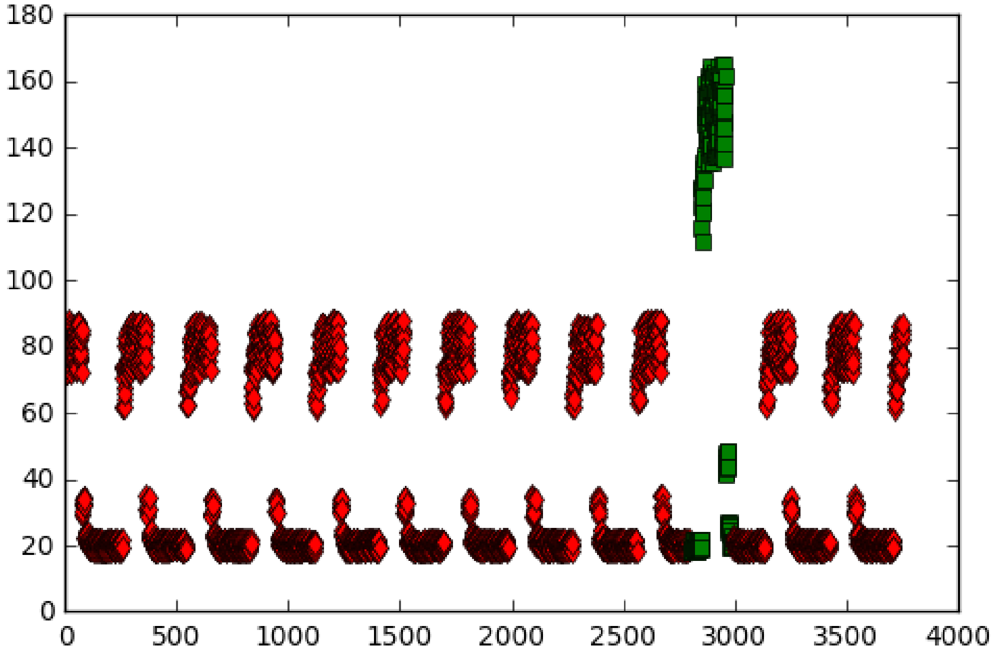
```
In [3]: def make_moa(data, size=20):
            """
            Returns a moving average for data, not a fast implementation
            """
            moa = []
            for i in range(0, len(data)):
                w = data[i:(i + size)]
                moa.append(sum(w) / len(w))
            return moa[:-size]
```

```
In [4]: def process_series(data, series):
            """
            Process data from a serie, marking outliers and plotting them

            A ideia é, primeiro eliminar o ruido com uma moving average,
            ela é executada duas vezes para suavizar contornos

            Depois o classificador IsolationForest é usado para encontrar
            grupos de outliers.

            Em seguida, esses grupos são processados para eliminar grupos
            muito pequenos.

            Por fim grupos grandes são conectados e o resultado é exibido.

            Para as séries apresentadas como só havia um outlier por arquivo
            conectei o primeiro e o último, para um arquivo com várias,
            precisaria definir um threshold para conectar as séries e juntar
            as que ficassem dentro do threshold.
            """
            # First make a moving average of the data, eliminating noise
            moa = np.array(make_moa(data, 80))
            moa = np.array(make_moa(moa, 200))
            moa = moa.reshape(-1, 1)
            # Then run IsolationForest to separate data
            c = IsolationForest(contamination=0.06, n_jobs=-1)
            c.fit(moa)
            pred = np.array(c.predict(moa))
            pred = pred.reshape(-1, 1)

            # Find outliers class
            c1s = sum([1 for x in pred if x == pred[0]])
            c2s = sum([1 for x in pred if x != pred[0]])
            if c1s < c2s:
                smallest_cluster = pred[0]
                biggest_cluster =  [x for x in pred if x != pred[0]][0]
            else:
                smallest_cluster = [x for x in pred if x != pred[0]][0]
                biggest_cluster = pred[0]

            # Remove small patches (less then 20 elements) of the small cluster
            pred_clean = []
            for i in range(len(pred)):
                if sum([1 for x in pred[i:i+20] if x == smallest_cluster]) < 20:
                    pred_clean.append(biggest_cluster)
                else:
                    pred_clean.append(smallest_cluster)

            # Connect outlier start and end
            for i in range(1, len(pred_clean)):
                if (smallest_cluster in pred_clean[i:] and
                    smallest_cluster in pred_clean[:i]):
                    pred_clean[i] = smallest_cluster

            # plot result
            plot_prediction(data[140:], pred_clean)

            return data[140:], pred_clean
```

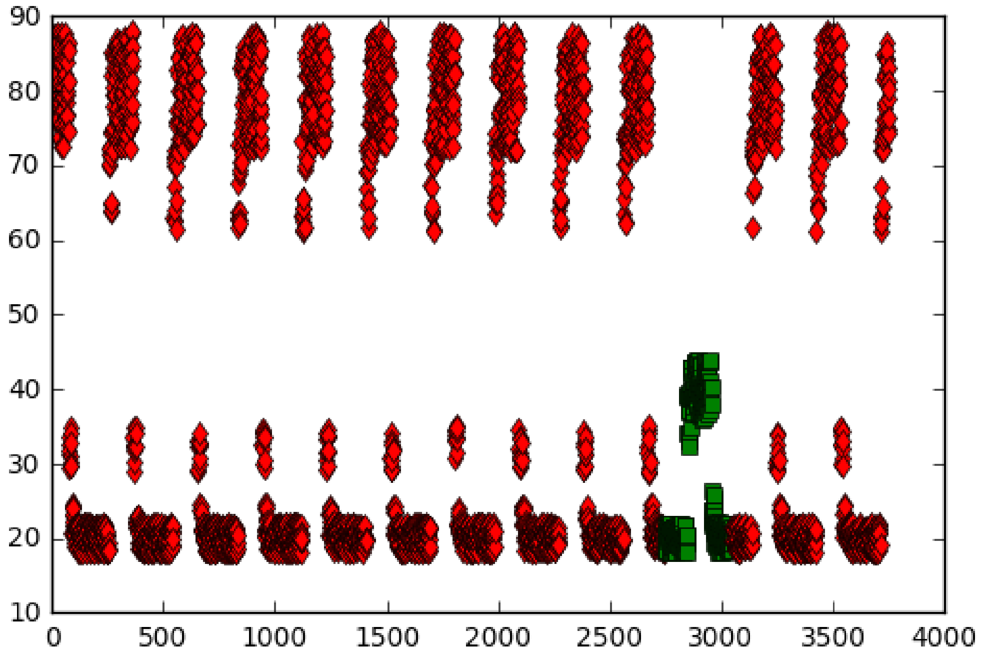
```
In [5]: def plot_prediction(serie, p):
            """ Plots series, uses p to decide colours """
            x = 0
            def f(x):
                if x == 1:
                    return "rd"
                else:
                    return "gs"
            pred = [f(x) for x in p]
            for point, color in zip(serie, pred):
                plt.plot(x, point, color)
                x = x + 1
            plt.show()
```

```
In [6]: data, series = read_data()
```

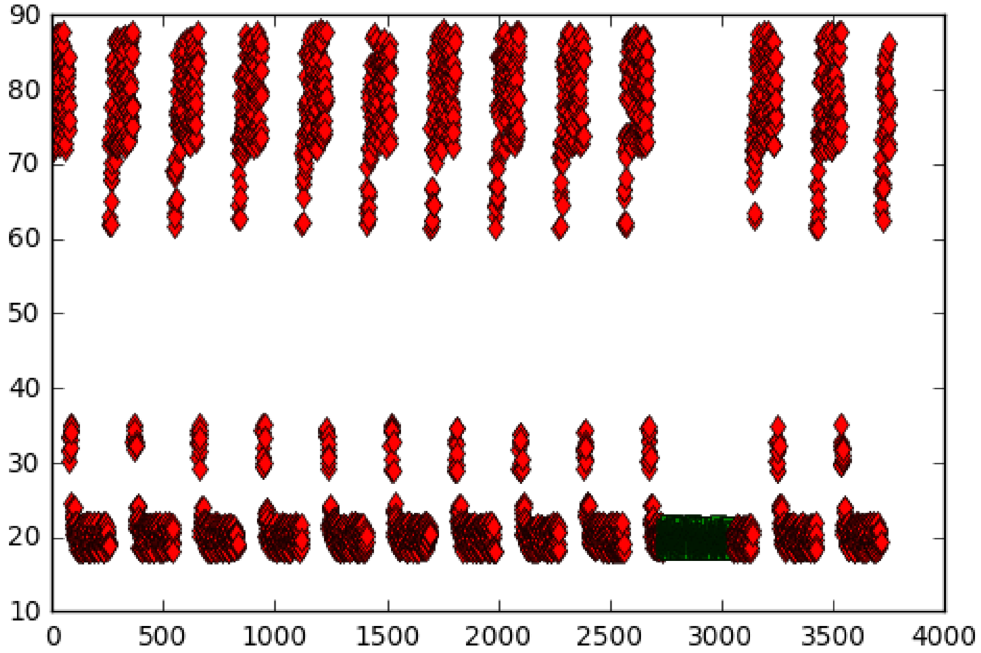
```
In [7]: dat, pred = process_series(data[0], series[0])
```



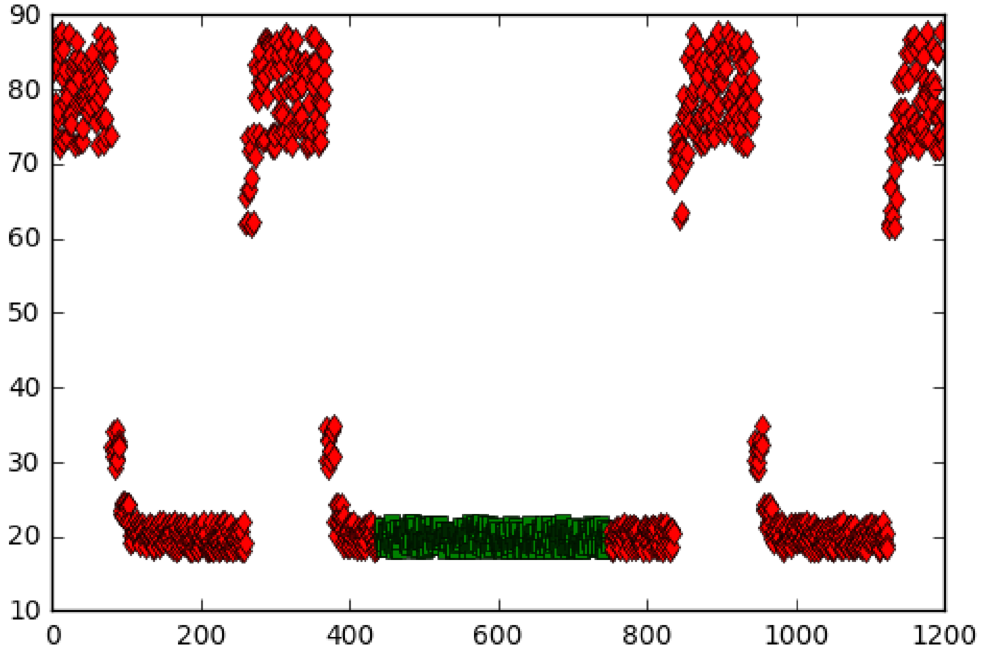
```
In [8]: dat, pred = process_series(data[1], series[1])
```



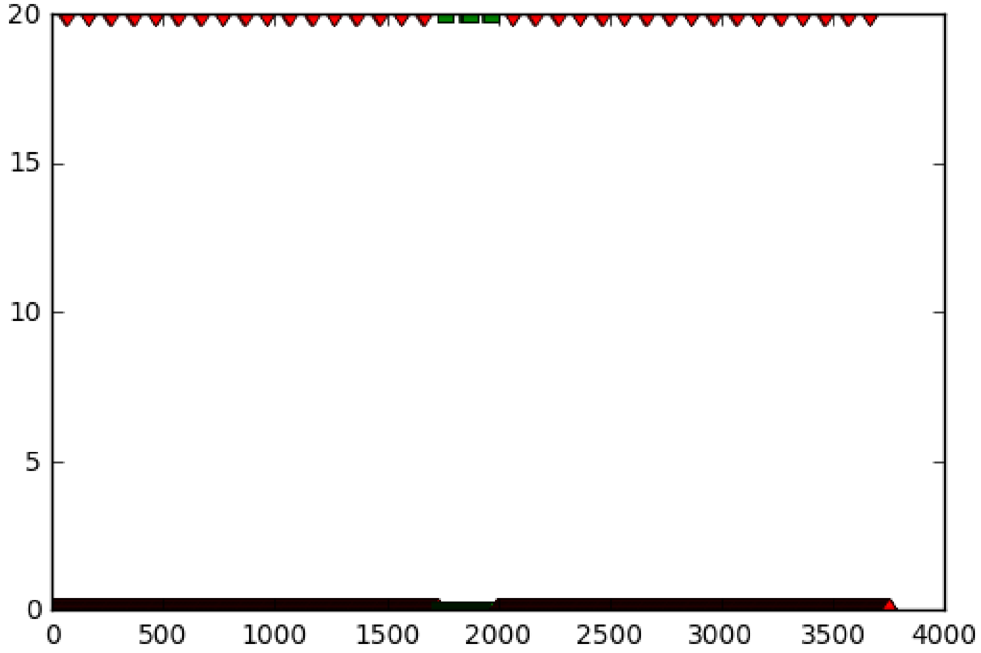
```
In [14]: dat, pred = process_series(data[2], series[2])
```



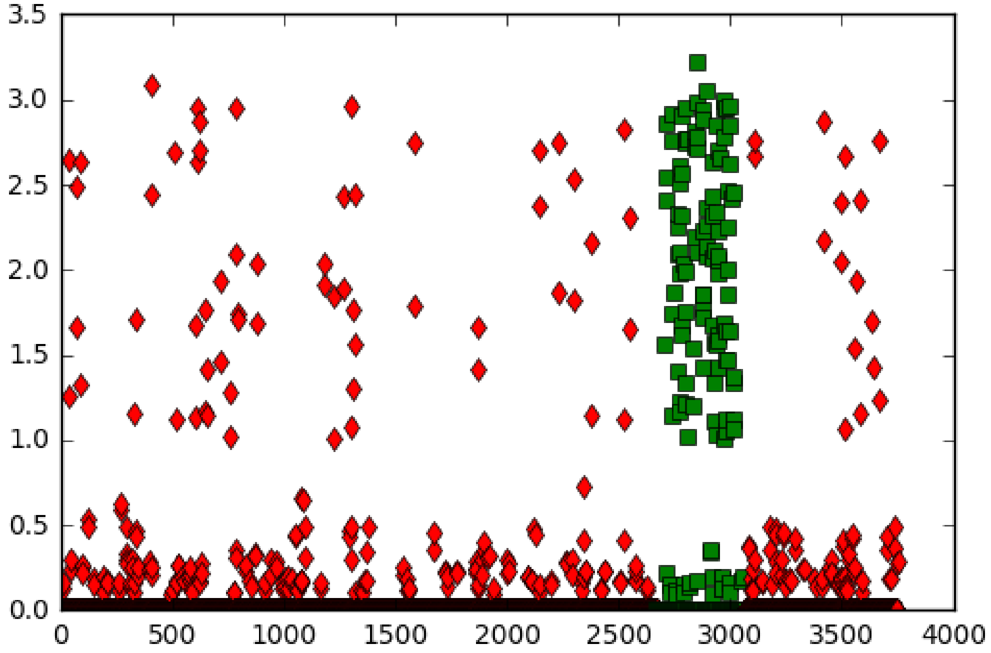
```
In [15]: # To help the visualization of the last graph, plotting a subsection
         plot_prediction(dat[2300:3500], pred[2300:3500])
```



```
In [11]: dat, pred = process_series(data[3], series[3])
```



```
In [16]: dat, pred = process_series(data[4], series[4])
```



3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
500
1000
1500
2000
2500
3000
3500
4000