異常値検知概要

異常検出は、異常値と呼ばれる、予想される動作に適合しない異常パターンを
特定するために使用される手法。

・ネットワーク侵入検知
（ネットワークトラフィックの 異常パターンを 識別）
・攻撃検知
・システムヘルスモニタリング（MRIスキャンで悪性腫瘍を発見）
・クレジットカード取引の不正検出
・携帯電話不正利用検知
・保険請求不正検知
・インサイダー取引検知
・画像診断：病巣検知
・産業機械・各種プラントの故障検知
・自動ニュース・タイムライン分類　新しい話題の検知

異常の分類

定義にいくつかの境界を確立することが重要。
異常は広く次のように分類できます。

1. 点異常 (point anomaly)
   データの値が他と離れすぎている。
   ビジネスユースケース
   「使用した金額」に基づいてクレジットカード詐欺を検出する

2. 文脈上の異常(contextual anomaly)
   文脈依存型異常とも。このタイプの異常は、時系列データでみることが多い。
   ビジネスユースケース
   ホリデーシーズンに毎日食べ物に100ドルを*使う*のはありうる？が、それ以外の場合は異常。

3. 集合的な異常(collective anpmaly)
   一連のデータインスタンスが集合的に異常を検出するのに役立つ。

新規性と外れ値の検出

外れ値(outlier)検出	データに異常値(ノイズ)が含まれている。 →意味のある信号からノイズを除去する。
新規性(novelty)検出	学習データに含まれていない新しい観測で観測されていないパターンの識別。 新しいトレンド検知 クリスマス期間の、YouTubeの新しいチャンネルの急激な関心。

新しい観測値が既存の観測値と同じ分布(inlier)に属しているのか、
異なるもの(outlier)であるのかを判断する必要がある。
↓
scikit-learn では 以下のステップでデータから教師なしで学習することが可能。

estimator.fit(X_train) # データから推定

これにより、

estimator.predict(X_test)

を実行すると、inlier は 1 とラベル付けされ、outlier は-1とラベル付けされる。

One-class SVM
One-class SVMは、ノベルティ検出のための決定関数を学習する教師なし学習アルゴリズム。
新しいデータをトレーニングセットと類似または異なるものとして分類する。
→後述。

検知結果の出力形式

- スコア
  異常度合いを示す連続的数値
- ラベル
  異常 or 正常

異常検知のアプローチ

- 単純移動平均による異常検出
- 密度に基づく異常検出

<u>属性値の型と手法</u>

- カテゴリ　categorical
- 連続値 numerical

統計的手法 → 連続値に適用可能
最近傍型手法 → 距離定義必要

単純移動平均による異常検出

単純移動平均（SMA, simple moving average）
データの不規則性を特定する最も簡単な方法は、平均、中央値、モード、および分位点を含む、分布の共通の統計的特性から逸脱したデータ点にフラグを立てること。
異常データポイントの定義が、平均値からの標準偏差を逸脱したものと扱う。
数学的には、n期単純移動平均を「ローパスフィルタ」と定義することもできる。

ローパスフィルタ（Low-pass filter: LPF、低域通過濾波器）
フィルタの一種で、なんらかの信号のうち、遮断周波数より低い周波数の成分は
ほとんど減衰させず、遮断周波数より高い周波数の成分を逓減させるフィルタである。

単純移動平均アプローチの問題点
単純なユースケースで異常を特定できますが、この手法がうまくいかない場合あり。

- 正常と異常の境界が不正確
  正常動作と異常動作の境界はしばしば正確ではない
  → データには異常動作と同様のノイズが含まれています。

- 異常の定義が変動する
  異常または正常の定義は頻繁に変動しうる。
  → 移動平均に基づく閾値は必ずしも適用されるとは限らない。

- 季節変動に対応できない
  パターンは季節性に基づく場合、データを複数の傾向に分解して季節性の変化を
  特定するなど、より洗練された方法が必要。

密度に基づく異常検出

密度に基づく異常検出は、k最近傍アルゴリズムに基づく。
仮定：通常のデータ点は密集した近傍で発生し、異常は遠く離れています。

最も近いデータ点のセットは、データの種類（カテゴリまたは数値）に依存するユークリッド距離 (Eucledian distance) または同様の尺度であり得るスコアを使用して評価される。
それらは大きく2つのアルゴリズムに分類することができます。

1. K-nearest neighbor
   k-NNは、Eucledian、Manhattan、Minkowski、またはHamming distanceなどの距離メトリックの類似性に基づいてデータを分類するために使用される。→参考1
   単純で非パラメトリックな遅延学習手法です。

   KNN のざっくり解説
   http://qiita.com/yshi12/items/26771139672d40a0be32

2. データの相対密度
   ローカルアウトライアファクタ （LOF）として知られる。
   到達可能距離と呼ばれる距離メトリックに基づいています。→参考2

参考1：距離のスコア比較

hamming distance

2進数表記された整数間のハミング距離とは、「2 つの数において値が異なる桁の数」である。
例： 000と 110 のハミング距離は 2 。

Euclidean/Manhattan/Minkowski distance

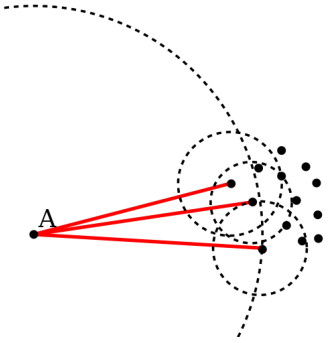
Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

Minkowski $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

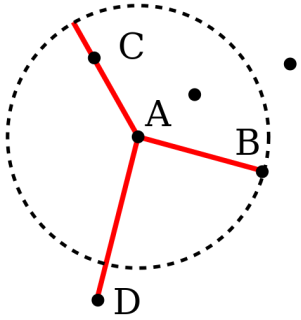
参考2：LOF(Local Outlier Factor)とは

データマイニング分野で利用される異常検知手法。局所的外れ値因子。
あるデータの密度がその他のデータの密度と比べて小さいかどうかを評価することで、
異常なデータを見つける。



LOFの基本的な考え方
点の局所密度をその近傍の密度と比較する。Aはその近隣よりもはるかに低い密度を有する。

局所密度は、ある点がその近傍から「到達」できる典型的な距離によって推定される。
LOFで使用される「**到達可能距離**」の定義は、クラスタ内でより安定した結果を生み出すための追加の手段です。



到達可能距離の図。
オブジェクトBとCは同じ到達可能距離（k = 3）を持つが、Dはk最近傍ではない。

クラスタリングによる異常検出

クラスタリングは教師なし学習の領域で最も一般的な概念の1つ。
仮定：類似しているデータ点は、ローカル重心からの距離によって決定されるように、
類似のグループまたはクラスタに属する傾向があります。

K-meansは、広く使用されているクラスタリングアルゴリズム。
データ点の「k」類似クラスターを作成する。これらのグループの外にあるデータは、
異常としてマークされる。

サポートベクターマシンベースの異常検出

サポートベクターマシン は通常教師あり学習に関連付けられていますが、教師なしの問題
（トレーニングデータにはラベルが付けられていない）として異常を特定するために使用できる
拡張機能（OneClassCVMなど）があります。

アルゴリズムは、トレーニングセットを使用して通常のデータインスタンスをクラスタリング
するためにソフト境界を学習し、テストインスタンスを使用して学習領域外の異常を識別
するように調整します。
ユースケースに応じて、異常検出器の出力は、ドメイン固有のしきい値またはテキストラベル
（バイナリ/マルチラベルなど）でのフィルタリングの数値スカラー値になります。

サンプル1：太陽黒点の変更データを利用した異常値検知

移動平均を使った簡単な異常検出の例。

サンプルデータセット
毎月の太陽黒点数の変動。ここから異常を特定することにトライする。
http://www-personal.umich.edu/~mejn/cp/data/sunspots.txt
wget -c -b http://www-personal.umich.edu/~mejn/cp/data/sunspots.txt

3,143の行があり、1749-1984年の間に収集された黒点に関する情報が含まれる。
太陽黒点は、太陽の表面の暗点として定義されます。太陽黒点の研究は、科学者が一定期間にわたって太陽の特性を理解するのに役立ちます。

離散直線畳み込みを用いた移動平均

畳み込みは、3つの関数を生成するために2つの関数で実行される数学的演算。
一方が反転してシフトされた後に数学的に、これは、2つの関数の積の積分として記述することができる
$\int_{-\infty}^{\infty} f(T) * g(t-T)dT$
=、F（T）は 、関心の量を含む入力関数である（例えば黒点は、時間のカウントT）。
g(t - T) は、量 t だけシフトされた重み付け関数。
このようにしてtが変化すると、異なる重みが入力関数 f(T) に割り当てられる。
f(T) は時間Tにおける太陽黒点の数を表す 。
g(t - T) は移動平均カーネル。

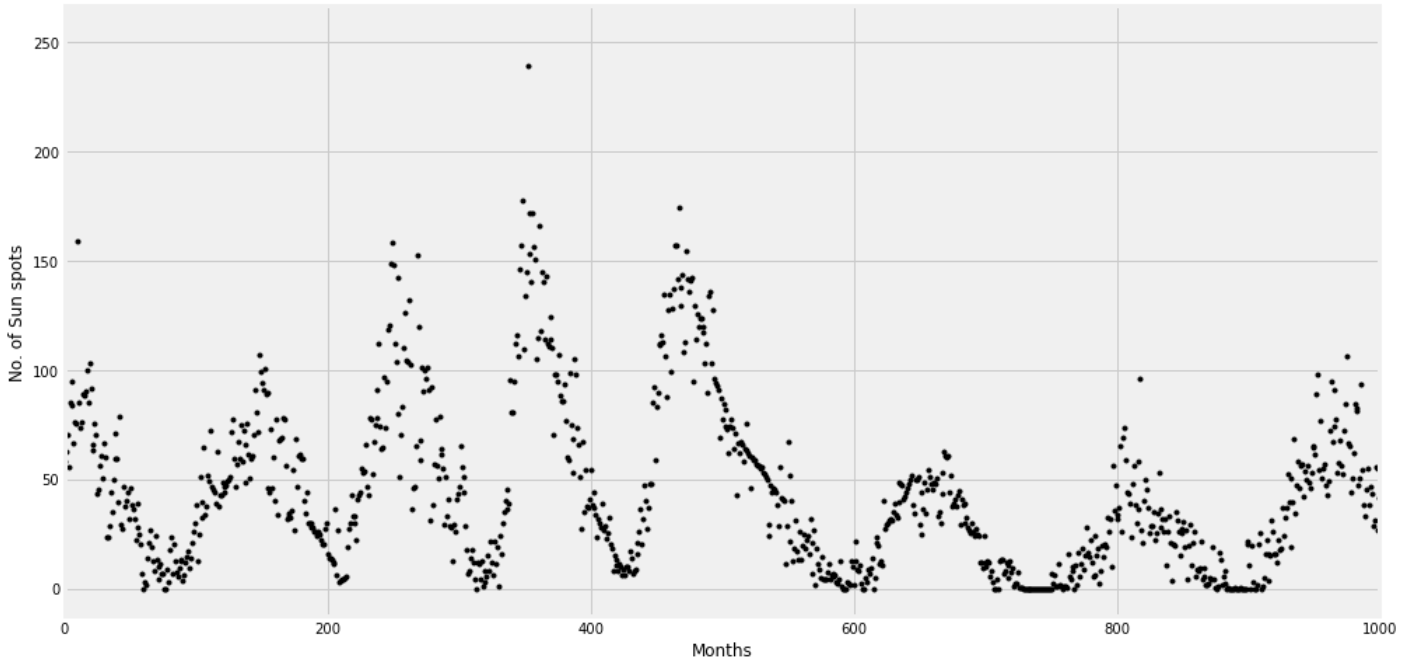
データのロード

```
---------------------------------------------------------------------
from __future__ import division
from itertools import count
import matplotlib.pyplot as plt
from numpy import linspace, loadtxt, ones, convolve
import numpy as np
import pandas as pd
import collections
from random import randint
from matplotlib import style
style.use('fivethirtyeight')
%matplotlib inline
# 1. Download sunspot dataset
#    Load the sunspot dataset as an Array
#    http://www-personal.umich.edu/~mejn/cp/data/sunspots.txt
data = loadtxt("/usr/local/zeppelin/work/anom/sunspots.txt", float)

# 2. View the data as a table
data_as_frame = pd.DataFrame(data, columns=['Months', 'SunSpots'])
data_as_frame.head()
---------------------------------------------------------------------
```

	Months	SunSpots
0	0.0	58.0
1	1.0	62.6
2	2.0	70.0
3	3.0	55.7
4	4.0	85.0

データのプロット

```
------------------------------------------------------------------------
x = data_as_frame['Months']
y = data_as_frame['SunSpots']

def plot_results1(x,y,text_xlabel="X Axis", text_ylabel="Y Axis"):
    plt.figure(figsize=(15, 8))
    plt.plot(x, y, "k.")
    plt.xlim(0, 1000)
    plt.xlabel(text_xlabel)
    plt.ylabel(text_ylabel)

    # add grid and lines and enable the plot
    plt.grid(True)
    plt.show()

plot_results1(x, y, text_xlabel = "Months", text_ylabel = "No. of Sun spots")

------------------------------------------------------------------------
```

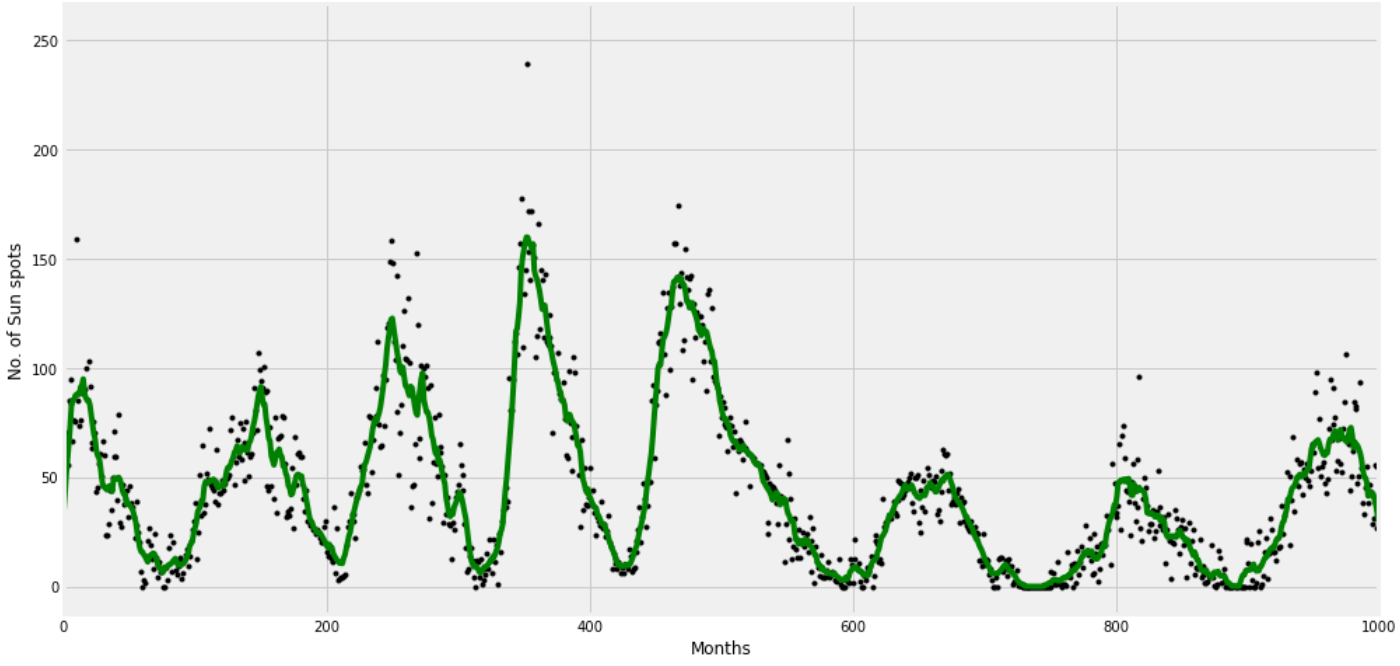


移動平均の計算

```
------------------------------------------------------------------------
# 2つの1次元シーケンスの離散線形畳み込みを使用して移動平均を計算します。
#
# Args:
#             data (pandas.Series): independent variable
#            window_size (int): rolling window size
# Returns:
#             ndarray of linear convolution
#
def moving_average(data, window_size):
    """
    References:
    ------------
    [1] Wikipedia, "Convolution", http://en.wikipedia.org/wiki/Convolution.
    [2] API Reference: https://docs.scipy.org/doc/numpy/reference/generated/numpy.convolve.html
    """
    window = np.ones(int(window_size))/float(window_size)
    #print("window:", window)
    return np.convolve(data, window, 'same')

------------------------------------------------------------------------
```

生データと移動平均のプロット

```
---------------------------------------------------------------------------
def plot_results2(x,y,text_xlabel="X Axis", text_ylabel="Y Axis",window_size=10):
    plt.figure(figsize=(15, 8))
    plt.plot(x, y, "k.")
    plt.xlim(0, 1000)
    plt.xlabel(text_xlabel)
    plt.ylabel(text_ylabel)

    # 移動平均データのプロット
    y_av = moving_average(y, window_size)
    plt.plot(x, y_av, color='green')

    # add grid and lines and enable the plot
    plt.grid(True)
    plt.show()

plot_results2(x, y, text_xlabel = "Months", text_ylabel = "No. of Sun spots",
              window_size=10)
---------------------------------------------------------------------------
```

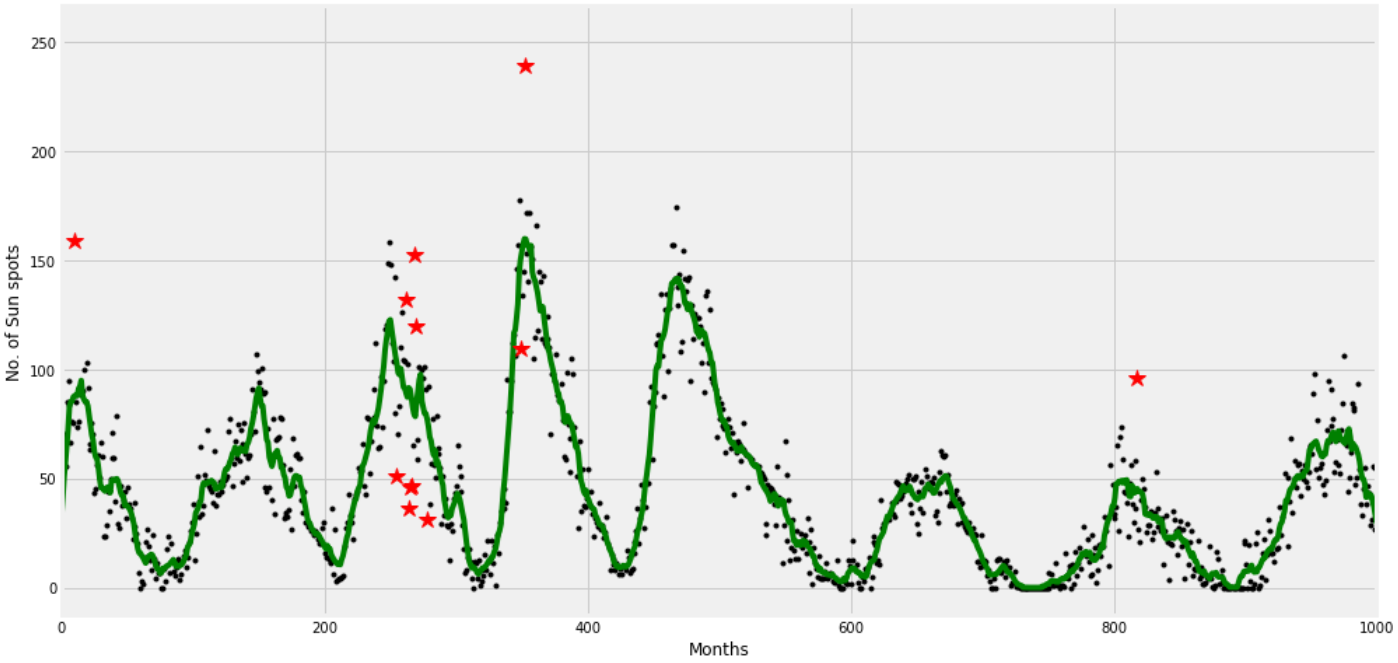


異常値の一覧取得

```
----------------------------------------------------------------------
# 異常値の一覧取得
#     残差分布の標準偏差を一定として
#     y > 移動平均 + (sigma*残差分布の標準偏差)) or y< 移動平均 - (sigma*残差分布の標準偏差 )
#     の場合に異常値とみなす
#
# Args:
#        y (pandas.Series): independent variable
#        window_size (int): rolling window size
#       sigma (int): value for standard deviation
# Returns:
#        a dict (dict of 'standard_deviation': int, 'anomalies_dict': (index: value))
#        containing information about the points indentified as anomalies
#
def explain_anomalies(y, window_size, sigma=1.0):
    avg = moving_average(y, window_size).tolist()
    residual = y - avg
    # Calculate the variation in the distribution of the residual
    std = round(np.std(residual), 3)

    anom = []
    for i in range(len(y)):
        y_i = y[i]
        avg_i = avg[i]
        if (y_i > avg_i + (sigma*std)) | (y_i < avg_i - (sigma*std)):
            anom = np.append(anom, [int(i), y_i])

    anom = anom.reshape(int(len(anom)/2), 2)
    return {'standard_deviation': std, 'anomalies_dict': anom}

# 異常値の一覧取得
#     残差分布の標準偏差を変動するものとして
#     y > 移動平均 + (sigma*残差分布の標準偏差)) or y< 移動平均 - (sigma*残差分布の標準偏差 )
#     の場合に異常値とみなす
def explain_anomalies_rolling_std(y, window_size, sigma=1.0):
    """
    Args:
    -----
        y (pandas.Series): independent variable
        window_size (int): rolling window size
        sigma (int): value for standard deviation

    Returns:
    --------
        a dict (dict of 'standard_deviation': int, 'anomalies_dict': (index: value))
        containing information about the points indentified as anomalies
    """
    avg = moving_average(y, window_size)
    avg_list = avg.tolist()
    residual = y - avg
    # Calculate the variation in the distribution of the residual
#    testing_std = pd.rolling_std(residual, window_size)
    testing_std = residual.rolling(window=window_size, center=False).std()
    testing_std_as_df = pd.DataFrame(testing_std)
    rolling_std = testing_std_as_df.replace(np.nan,
                  testing_std_as_df.iloc[window_size - 1]).round(3).iloc[:,0].tolist()
    std = np.std(residual)
    std = round(std, 3)

    anom = []
    for i in range(len(y)):
        y_i = y[i]
        avg_i = avg[i]
        std_i = rolling_std[i]
        if (y_i > avg_i + (sigma*std_i)) | (y_i < avg_i - (sigma*std_i)):
            anom = np.append(anom, [int(i), y_i])

    anom = anom.reshape(int(len(anom)/2), 2)
    return {'stationary standard_deviation': std, 'anomalies_dict': anom}

----------------------------------------------------------------------
```

生データと移動平均、異常値のプロット

```
----------------------------------------------------------------------
# データのプロット
#
#    Args:
#    -----
#        x (pandas.Series): dependent variable
#        y (pandas.Series): independent variable
#        window_size (int): rolling window size
#        sigma_value (int): value for standard deviation
#        text_xlabel (str): label for annotating the X Axis
#        text_ylabel (str): label for annotatin the Y Axis
#        applying_rolling_std (boolean): True/False for using rolling vs stationary standard
deviation
def plot_results(x, y, window_size, sigma_value=1,
                 text_xlabel="X Axis", text_ylabel="Y Axis", rolling_std=False):
    plt.figure(figsize=(15, 8))
    plt.xlim(0, 1000)
    plt.xlabel(text_xlabel)
    plt.ylabel(text_ylabel)

    # 生データのプロット
    plt.plot(x, y, "k.")

    # 移動平均データのプロット
    y_av = moving_average(y, window_size)
    plt.plot(x, y_av, color='green')

    # Query for the anomalies and plot the same
    events = {}
    if rolling_std:
        events = explain_anomalies_rolling_std(y, window_size=window_size, sigma=sigma_value)
    else:
        events = explain_anomalies(y, window_size=window_size, sigma=sigma_value)

    x_anomaly = np.fromiter(events['anomalies_dict'][:,0], dtype=int,
                            count=len(events['anomalies_dict']))
    y_anomaly = np.fromiter(events['anomalies_dict'][:,1], dtype=float,
                            count=len(events['anomalies_dict']))
    plt.plot(x_anomaly, y_anomaly, "r*", markersize=12)

    # add grid and lines and enable the plot
    plt.grid(True)
    plt.show()

# 4. Lets play with the functions
x = data_as_frame['Months']
Y = data_as_frame['SunSpots']

# plot the results
rolling_std = False
plot_results(x, y=Y, window_size=10, sigma_value=3,
             text_xlabel="Months", text_ylabel="No. of Sun spots",
            rolling_std=rolling_std)

# Display the anomaly dict
#print("Information about the anomalies model:{}".format(events))
----------------------------------------------------------------------
```


No. of Sun spots
Months
0
50
100
150
200
250
0
200
400
600
800
1000

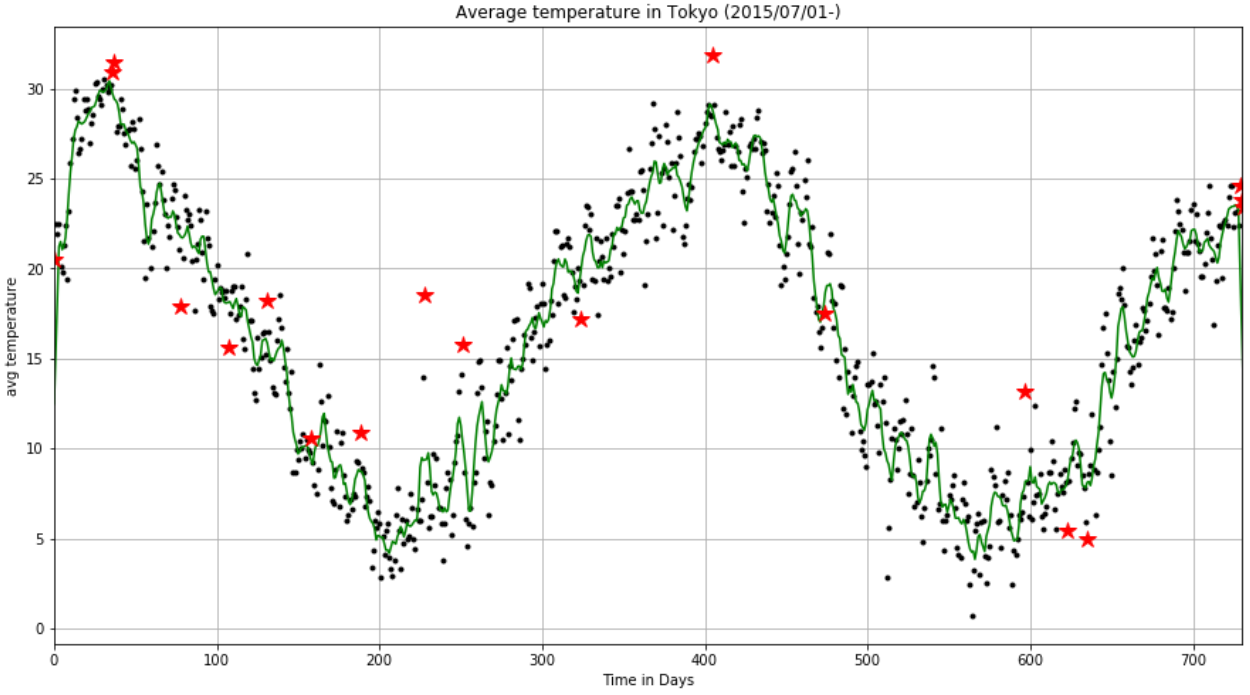
サンプル２：東京の気温データを利用した異常値検知

東京の過去２年の気温データを用いた場合。

```
----------------------------------------------------------------------
import numpy as np
from itertools import count
import pandas as pd
import matplotlib.pyplot as plt

# 1. Download temperature dataset
"""
気象庁が過去の色んなデータを公開しているので、ここからcsvをダウンロード
http://www.data.jma.go.jp/gmd/risk/obsdl/index.php

データは東京の2015/07/01 から2017/01/01 の平均、最高、最低気温
"""
data = pd.DataFrame.from_csv('/usr/local/zeppelin/work/temp-tokyo2.csv', header=None, sep=',')
#print(data)

# 2. View the data as a table
data_as_frame = data.rename(columns={1: 'avg', 2:'high', 3:'low'}, inplace=False)
#print(data_as_frame.head(10))
#print("data size:", len(data_as_frame))
----------------------------------------------------------------------


----------------------------------------------------------------------
# 2つの1次元シーケンスの離散線形畳み込みを使用して移動平均を計算します。
#
# Args:
#             data (pandas.Series): independent variable
#           window_size (int): rolling window size
# Returns:
#             ndarray of linear convolution
#
def moving_average(data, window_size):
    """
    References:
    ------------
    [1] Wikipedia, "Convolution", http://en.wikipedia.org/wiki/Convolution.
    [2] API Reference: https://docs.scipy.org/doc/numpy/reference/generated/numpy.convolve.html
    """
    window = np.ones(int(window_size))/float(window_size)
    #print("window:", window)
    return np.convolve(data, window, 'same')

# 異常値の一覧取得
#     残差分布の標準偏差を一定として
#     y > 移動平均 + (sigma*残差分布の標準偏差)) or y< 移動平均 - (sigma*残差分布の標準偏差 )
#     の場合に異常値とみなす
#
# Args:
#         y (pandas.Series): independent variable
#         window_size (int): rolling window size
#       sigma (int): value for standard deviation
# Returns:
#         a dict (dict of 'standard_deviation': int, 'anomalies_dict': (index: value))
#         containing information about the points indentified as anomalies
#
def explain_anomalies(y, window_size, sigma=1.0):
    avg = moving_average(y, window_size).tolist()
    residual = y - avg
    # Calculate the variation in the distribution of the residual
    std = round(np.std(residual), 3)

    anom = []
    for i in range(len(y)):
        y_i = y[i]
        avg_i = avg[i]
        if (y_i > avg_i + (sigma*std)) | (y_i < avg_i - (sigma*std)):
            anom = np.append(anom, [int(i), y_i])
```

```
    anom = anom.reshape(int(len(anom)/2), 2)
    return {'standard_deviation': std, 'anomalies_dict': anom}

# 異常値の一覧取得
#     残差分布の標準偏差を変動するものとして
#     y > 移動平均 + (sigma*残差分布の標準偏差)) or y< 移動平均 - (sigma*残差分布の標準偏差 )
#     の場合に異常値とみなす
def explain_anomalies_rolling_std(y, window_size, sigma=1.0):
    """
    Args:
    -----
        y (pandas.Series): independent variable
        window_size (int): rolling window size
        sigma (int): value for standard deviation

    Returns:
    --------
        a dict (dict of 'standard_deviation': int, 'anomalies_dict': (index: value))
        containing information about the points indentified as anomalies
    """
    avg = moving_average(y, window_size)
    avg_list = avg.tolist()
    residual = y - avg
    # Calculate the variation in the distribution of the residual
    testing_std = pd.rolling_std(residual, window_size)
    #testing_std = pd.rolling_std(residual, window=window_size).values.flatten()
#     testing_std = residual.rolling(window=window_size, center=False).std()
    testing_std_as_df = pd.DataFrame(testing_std)
    rolling_std = testing_std_as_df.replace(np.nan,
                  testing_std_as_df.iloc[window_size - 1]).round(3).iloc[:,0].tolist()
    std = np.std(residual)
    std = round(std, 3)

    anom = []
    for i in range(len(y)):
        y_i = y[i]
        avg_i = avg[i]
        std_i = rolling_std[i]
        if (y_i > avg_i + (sigma*std_i)) | (y_i < avg_i - (sigma*std_i)):
            anom = np.append(anom, [int(i), y_i])

    anom = anom.reshape(int(len(anom)/2), 2)
    return {'stationary standard_deviation': std, 'anomalies_dict': anom}


# データのプロット
#
#    Args:
#    -----
#        x (pandas.Series): dependent variable
#        y (pandas.Series): independent variable
#        window_size (int): rolling window size
#        sigma_value (int): value for standard deviation
#        text_xlabel (str): label for annotating the X Axis
#        text_ylabel (str): label for annotatin the Y Axis
#        applying_rolling_std (boolean): True/False for using rolling vs stationary standard
deviation
def plot_results(x, y, window_size, sigma_value=1,
                 text_xlabel="X Axis", text_ylabel="Y Axis",
                 title="title", rolling_std=False):
    plt.figure(figsize=(15, 8))
    plt.title(title)
    plt.xlim(0, 730)
    plt.xlabel(text_xlabel)
    plt.ylabel(text_ylabel)

    # 生データのプロット
    plt.plot(x, y, "k.")

    # 移動平均データのプロット
    y_av = moving_average(y, window_size)
    plt.plot(x, y_av, color='green')

    # Query for the anomalies and plot the same
```

```
    events = {}
    if rolling_std:
        events = explain_anomalies_rolling_std(y, window_size=window_size, sigma=sigma_value)
    else:
        events = explain_anomalies(y, window_size=window_size, sigma=sigma_value)

    x_anomaly = np.fromiter(events['anomalies_dict'][:,0], dtype=int,
                            count=len(events['anomalies_dict']))
    y_anomaly = np.fromiter(events['anomalies_dict'][:,1], dtype=float,
                            count=len(events['anomalies_dict']))
    plt.plot(x_anomaly, y_anomaly, "r*", markersize=12)
    # add grid and lines and enable the plot
    plt.grid(True)
    plt.show()

# Lets play
x1 = np.arange(len(data_as_frame))
y1 = np.array(data_as_frame[['avg']].values.flatten())
rolling_std = True
plot_results(x1, y1, window_size=7, title="Average temperature in Tokyo (2015/07/01-)",
                     sigma_value=2, text_xlabel="Time in Days", text_ylabel="avg temperature",
rolling_std=rolling_std)

------------------------------------------------------------------------
```



サンプル３：IsolationForestを利用したノベルティ検知

工事中

参考文献

Minkowski distance
https://en.wikipedia.org/wiki/Minkowski_distance

Hamming distance
https://en.wikipedia.org/wiki/Hamming_distance

K-nearest neighbor
http://www.scholarpedia.org/article/K-nearest_neighbor

KNN のざっくり解説
http://qiita.com/yshi12/items/26771139672d40a0be32

Local outlier factor
https://en.wikipedia.org/wiki/Local_outlier_factor

Local outlier factor の元論文
LOF: Identifying Density-Based Local Outliers
http://www.dbs.ifi.lmu.de/Publikationen/Papers/LOF.pdf

Local Outlier Factorによる異常検知
http://qiita.com/saigyo-k/items/5bfff4403b0fb171beb4

Introduction to K-means Clustering
https://goo.gl/MM9tBV

Novelty and Outlier Detection
http://scikit-learn.org/0.18/modules/outlier_detection.html

One-class SVM with non-linear kernel (RBF)
http://scikit-learn.org/0.18/auto_examples/svm/plot_oneclass.html#sphx-glr-auto-examples-svm-plot-oneclass-py

IsolationForest example
http://scikit-learn.org/0.18/auto_examples/ensemble/plot_isolation_forest.html#sphx-glr-auto-examples-ensemble-plot-isolation-forest-py

Robust covariance estimation and Mahalanobis distances relevance
http://scikit-learn.org/0.18/auto_examples/covariance/plot_mahalanobis_distances.html#sphx-glr-auto-examples-covariance-plot-mahalanobis-distances-py