Anomaly Detection

Introduction: Anomaly Detection

Anomaly detection is a technique used to identify unusual patterns that do not conform to expected behavior, called outliers. It has many applications in business, from intrusion detection (identifying strange patterns in network traffic that could signal a hack) to system health monitoring (spotting a malignant tumor in an MRI scan), and from fraud detection in credit card transactions to fault detection in operating environments.

This overview will cover several methods of detecting anomalies, as well as how to build a detector in Python using simple moving average (SMA) or low-pass filter.

What Are Anomalies?

Before getting started, it is important to establish some boundaries on the definition of an anomaly. Anomalies can be broadly categorized as:

1. **Point anomalies:** A single instance of data is anomalous if it's too far off from the rest. *Business use case:* Detecting credit card fraud based on "amount spent."
2. **Contextual anomalies:** The abnormality is context specific. This type of anomaly is common in time-series data. *Business use case:* Spending $100 on food every day during the holiday season is normal, but may be odd otherwise.
3. **Collective anomalies:** A set of data instances collectively helps in detecting anomalies. *Business use case:* Someone is trying to copy data form a remote machine to a local host unexpectedly, an anomaly that would be flagged as a potential cyber attack.

Anomaly detection is similar to — but not entirely the same as — noise removal and novelty detection. **Novelty detection** is concerned with identifying an unobserved pattern in new observations not included in training data — like a sudden interest in a new channel on YouTube during Christmas, for instance. **Noise removal** (NR) is the process of immunizing analysis from the occurrence of unwanted observations; in other words, removing noise from an otherwise meaningful signal.

Anomaly Detection Techniques

Simple Statistical Methods

The simplest approach to identifying irregularities in data is to flag the data points that deviate from common statistical properties of a distribution, including mean, median, mode, and quantiles. Let's say the definition of an anomalous data point is one that deviates by a certain standard deviation from the mean. Traversing mean over time-series data isn't exactly trivial, as it's not static. You would need a rolling window to compute the average across the data points. Technically, this is called a rolling average or a moving average, and it's intended to smooth short-term fluctuations and highlight long-term ones. Mathematically, an n-period simple moving average can also be defined as a "low pass filter." (A Kalman filter is a more sophisticated version of this metric; you can find a very intuitive explanation of it here.)

Challenges

The low pass filter allows you to identify anomalies in simple use cases, but there are certain situations where this technique won't work. Here are a few:

- The data contains noise which might be similar to abnormal behavior, because the boundary between normal and abnormal behavior is often not precise.
- The definition of abnormal or normal may frequently change, as malicious adversaries constantly adapt themselves. Therefore, the threshold based on moving average may not always apply.
- The pattern is based on seasonality. This involves more sophisticated methods, such as decomposing the data into multiple trends in order to identify the change in seasonality.

Machine Learning-Based Approaches

Below is a brief overview of popular machine learning-based techniques for anomaly detection.

Density-Based Anomaly Detection

Density-based anomaly detection is based on the k-nearest neighbors algorithm.

Assumption: Normal data points occur around a dense neighborhood and abnormalities are far away.

The nearest set of data points are evaluated using a score, which could be Eucledian distance or a similar measure dependent on the type of the data (categorical or numerical). They could be broadly classified into two algorithms:

1. K-nearest neighbor: k-NN is a simple, non-parametric lazy learning technique used to classify data based on similarities in distance metrics such as Eucledian, Manhattan, Minkowski, or Hamming distance.
2. Relative density of data: This is better known as local outlier factor (LOF). This concept is based on a distance metric called reachability distance.

Clustering-Based Anomaly Detection

Clustering is one of the most popular concepts in the domain of unsupervised learning.

Assumption: Data points that are similar tend to belong to similar groups or clusters, as determined by their distance from local centroids.

K-means is a widely used clustering algorithm. It creates 'k' similar clusters of data points. Data instances that fall outside of these groups could potentially be marked as anomalies.

Support Vector Machine-Based Anomaly Detection

A support vector machine is another effective technique for detecting anomalies. A SVM is typically associated with supervised learning, but there are extensions (OneClassCVM, for instance) that can be used to identify anomalies as an unsupervised problems (in which training data are not labeled). The algorithm learns a soft boundary in order to cluster the normal data instances using the training set, and then, using the testing instance, it tunes itself to identify the abnormalities that fall outside the learned region.

Depending on the use case, the output of an anomaly detector could be numeric scalar values for filtering on domain-specific thresholds or textual labels (such as binary/multi labels).

Building a Simple Detection Solution Using a Low-Pass Filter

In this section, we will focus on building a simple anomaly-detection package using moving average to identify anomalies in the number of sunspots per month in a sample dataset, which can be downloaded here using the following command:

```
wget -c -b http://www-personal.umich.edu/~mejn/cp/data/sunspots.txt
```

The file has 3,143 rows, which contain information about sunspots collected between the years 1749-1984. Sunspots are defined as dark spots on the surface of the sun. The study of sunspots helps scientists understand the sun's properties over a period of time; in particular, its magnetic properties.

Moving Average Using Discrete Linear Convolution

Convolution is a mathematical operation that is performed on two functions to produce a third function. Mathematically, it could be described as the integral of the product of two functions, after one is reversed and shifted: $f * g(t) = \int_{-\infty}^{\infty} f(T) * g(t - T)dT$, where *f(T)* is an input function containing the quantity of interest (e.g. sunspot count at time *T*). *g(t —T)* is the weighting function shifted by an amount *t*. This way as *t* changes, different weights are assigned to the input function *f(T)*. In our case, *f(T)* represents the sunspot counts at time T. *g(t —T)* is the moving average kernel.

```
from __future__ import division
from itertools import izip, 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 and upload the same to dataset directory
#    Load the sunspot dataset as an Array
!mkdir -p dataset
!wget -c -b http://www-personal.umich.edu/~mejn/cp/data/sunspots.txt -P dataset
data = loadtxt("dataset/sunspots.txt", float)

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

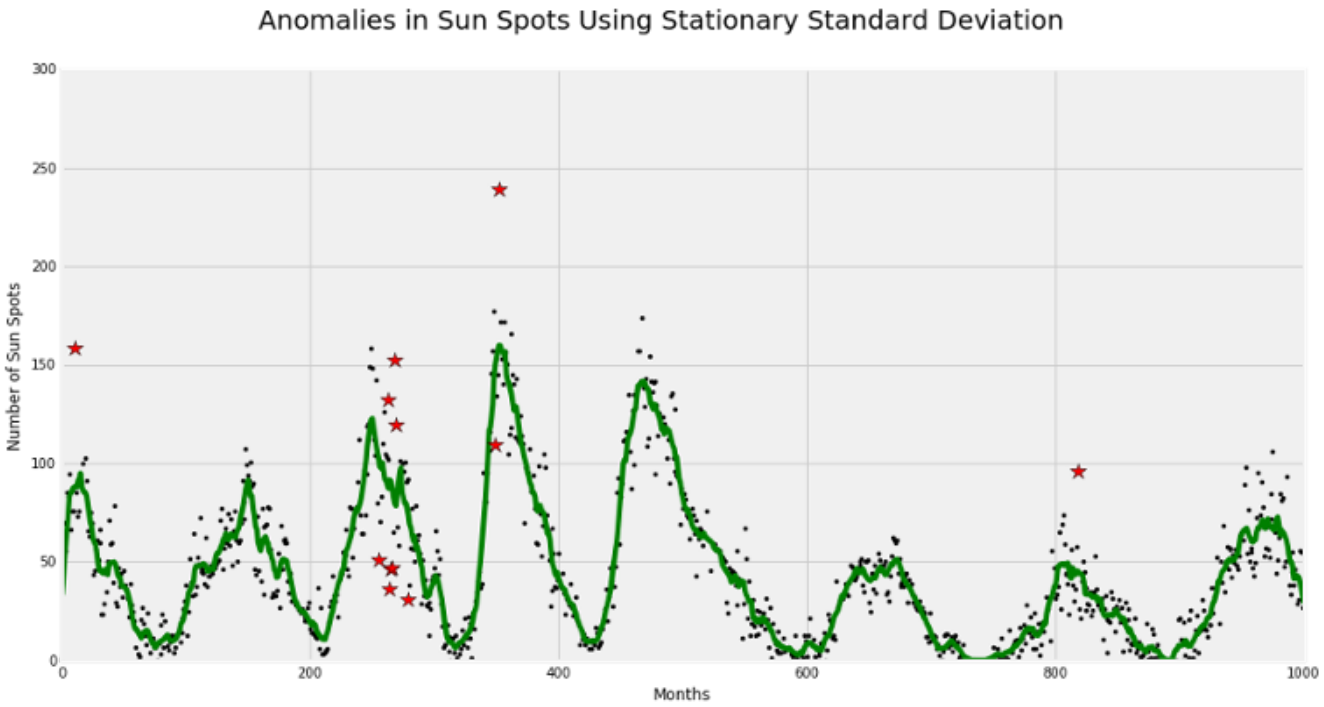
	Months	SunSpots
0	0.0	58.0
1	1.0	62.6
2	2.0	70.0
3	3.0	55.7
4	4.0	85.0

```
# 3. Lets define some use-case specific UDF(User Defined Functions)

def moving_average(data, window_size):
    """ Computes moving average using discrete linear convolution of two one dimensional
sequences.
    Args:
    -----
            data (pandas.Series): independent variable
            window_size (int): rolling window size

    Returns:
    --------
            ndarray of linear convolution

    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)
    return np.convolve(data, window, 'same')


def explain_anomalies(y, window_size, sigma=1.0):
  """ Helps in exploring the anamolies using stationary standard deviation
    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).tolist()
    residual = y - avg
    # Calculate the variation in the distribution of the residual
    std = np.std(residual)
    return {'standard_deviation': round(std, 3),
            'anomalies_dict': collections.OrderedDict([(index, y_i) for
                                                       index, y_i, avg_i in izip(count(),
y, avg)
              if (y_i > avg_i + (sigma*std)) | (y_i < avg_i - (sigma*std))])}


def explain_anomalies_rolling_std(y, window_size, sigma=1.0):
    """ Helps in exploring the anamolies using rolling standard deviation
    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_as_df = pd.DataFrame(testing_std)
```

```
    rolling_std = testing_std_as_df.replace(np.nan,
                                    testing_std_as_df.ix[window_size -
1]).round(3).iloc[:,0].tolist()
    std = np.std(residual)
    return {'stationary standard_deviation': round(std, 3),
            'anomalies_dict': collections.OrderedDict([(index, y_i)
                                                       for index, y_i, avg_i, rs_i in
izip(count(),

  y, avg_list, rolling_std)
             if (y_i > avg_i + (sigma * rs_i)) | (y_i < avg_i - (sigma * rs_i))])}


# This function is repsonsible for displaying how the function performs on the given
dataset.
def plot_results(x, y, window_size, sigma_value=1,
                 text_xlabel="X Axis", text_ylabel="Y Axis", applying_rolling_std=False):
    """ Helps in generating the plot and flagging the anamolies.
        Supports both moving and stationary standard deviation. Use the
'applying_rolling_std' to switch
        between the two.
    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
    """
    plt.figure(figsize=(15, 8))
    plt.plot(x, y, "k.")
    y_av = moving_average(y, window_size)
    plt.plot(x, y_av, color='green')
    plt.xlim(0, 1000)
    plt.xlabel(text_xlabel)
    plt.ylabel(text_ylabel)

    # Query for the anomalies and plot the same
    events = {}
    if applying_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'].iterkeys(), dtype=int,
count=len(events['anomalies_dict']))
    y_anomaly = np.fromiter(events['anomalies_dict'].itervalues(), dtype=float,
                                           count=len(events['anomalies_dict']))
    plt.plot(x_anomaly, y_anomaly, "r*", markersize=12)
```

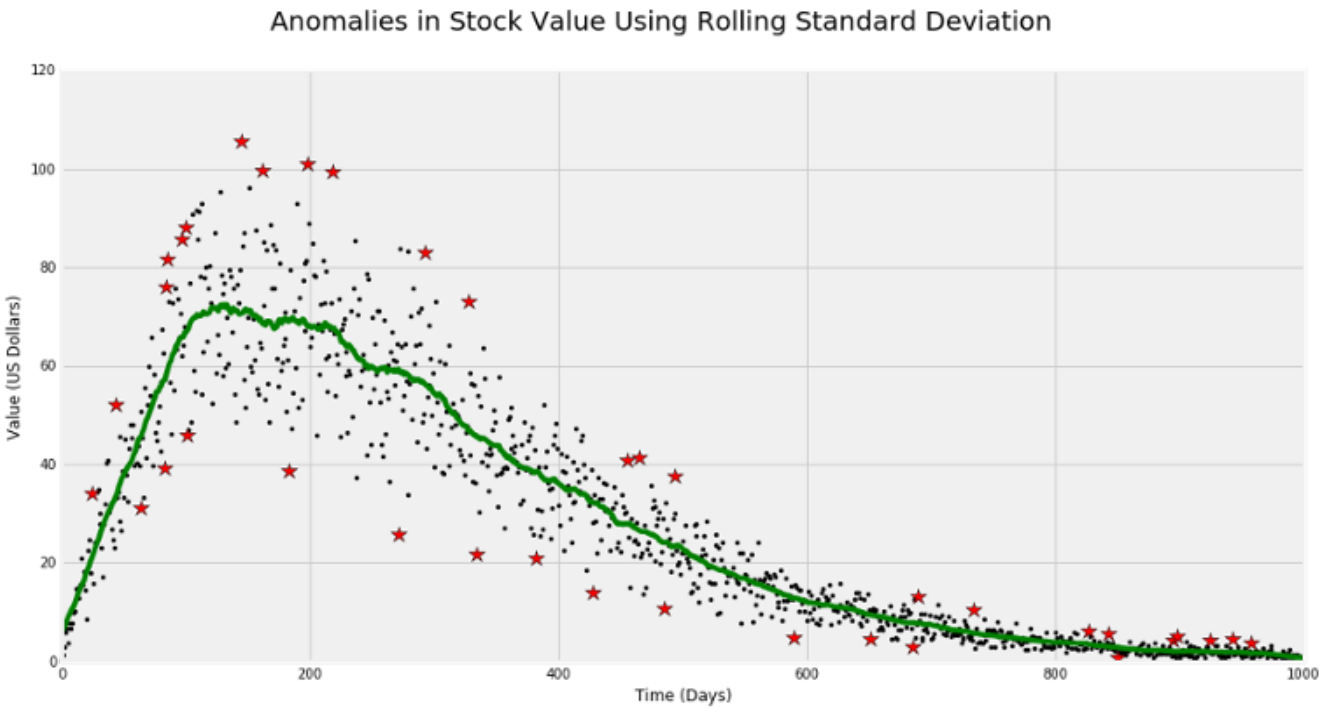
```
    # add grid and lines and enable the plot
    plt.grid(True)
    plt.show()
# 4. Lets play with the functions
x = data_as_frame['Months']
Y = data_as_frame['SunSpots']

# plot the results
plot_results(x, y=Y, window_size=10, text_xlabel="Months", sigma_value=3,
             text_ylabel="No. of Sun spots")
events = explain_anomalies(y, window_size=5, sigma=3)

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

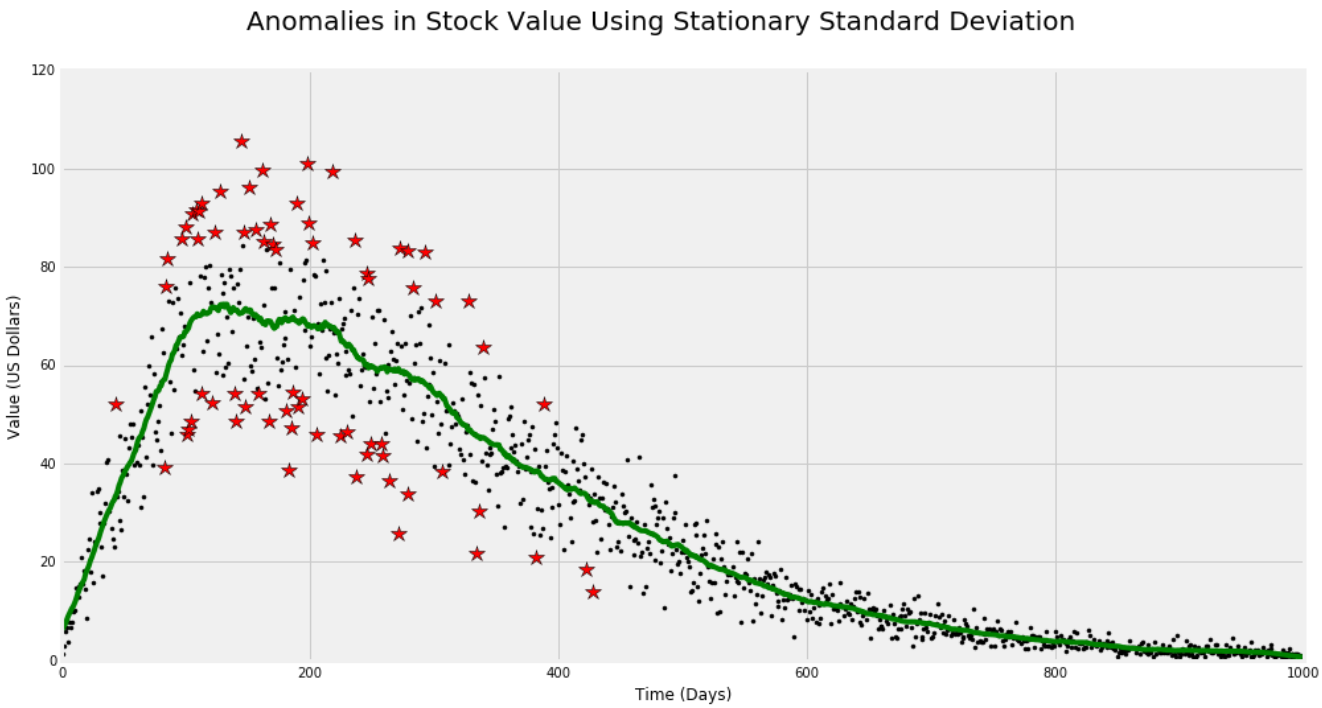


Let's see if the above anomaly detection function could be used for another use case. Let's assume that we generate a random dataset that hypothetically relates to Company A's stock value over a period of time. The x axis represents time in days (since 2013) and the y axis represents the value of the stock in dollars.

```
# Convenience function to add noise
def noise(yval):
    """ Helper function to generate random points """
    np.random.seed(0)
    return 0.2*np.asarray(yval)*np.random.normal(size=len(yval))

# Generate a random dataset
def generate_random_dataset(size_of_array=1000, random_state=0):
    """ Helps in generating a random dataset which has a normal distribution
    Args:
    -----
        size_of_array (int): number of data points
```

```
        random_state (int): to initialize a random state

    Returns:
    --------
        a list of data points for dependent variable, pandas.Series of independent
variable
    """
    np.random.seed(random_state)
    y = np.random.normal(0, 0.5, size_of_array)
    x = range(0, size_of_array)
    y_new = [y_i + index**((size_of_array - index)/size_of_array) +  noise()
     for index, y_i in izip(count(), y)]
    return x, pd.Series(y_new)


# Lets play
x1, y1 = generate_random_dataset()
# Using stationary standard deviation over a continuous sample replicating
plot_results(x1, y1, window_size=12, title_for_plot="Statinoary Standard Deviation",
                    sigma_value=2, text_xlabel="Time in Days", text_ylabel="Value in $")

# using rolling standard deviation for
x1, y1 = generate_random_dataset()
plot_results(x1, y1, window_size=50, title_for_plot="Using rolling standard deviation",
             sigma_value=2, text_xlabel="Time in Days", text_ylabel="Value in $",
applying_rolling_std=True)
```





LEARN DATA SCIENCE, PYTHON, MACHINE LEARNING

Introduction to Anomaly Detection

Author: Pramit Choudhary Posted on February 15, 2017

Prerequisites

Experience with the specific topic: Novice

Professional experience: No industry experience

This overview is intended for beginners in the fields of data science and machine learning. Almost no formal professional experience is needed to follow along, but the reader should have some basic knowledge of calculus (specifically integrals), the programming language Python, functional programming, and machine learning.

Introduction: Anomaly Detection

Anomaly detection is a technique used to identify unusual patterns that do not conform to expected behavior, called outliers. It has many applications in business, from intrusion detection (identifying strange patterns in network traffic that could signal a hack) to system health monitoring (spotting a malignant tumor in an MRI scan), and from fraud detection in credit card transactions to fault detection in operating environments.

This overview will cover several methods of detecting anomalies, as well as how to build a detector in Python using simple moving average (SMA) or low-pass filter.

What Are Anomalies?

Before getting started, it is important to establish some boundaries on the definition of an anomaly. Anomalies can be broadly categorized as:

1. **Point anomalies:** A single instance of data is anomalous if it's too far off from the rest. *Business use case:* Detecting credit card fraud based on "amount spent."
2. **Contextual anomalies:** The abnormality is context specific. This type of anomaly is common in time-series data. *Business use case:* Spending $100 on food every day during the holiday season is normal, but may be odd otherwise.
3. **Collective anomalies:** A set of data instances collectively helps in detecting anomalies. *Business use case:* Someone is trying to copy data form a remote machine to a local host unexpectedly, an anomaly that would be flagged as a potential cyber attack.

Anomaly detection is similar to — but not entirely the same as — noise removal and novelty detection. **Novelty detection** is concerned with identifying an unobserved pattern in new observations not included in training data — like a sudden interest in a new channel on YouTube during Christmas, for instance. **Noise removal** (NR) is the process of immunizing analysis from the occurrence of unwanted observations; in other words, removing noise from an otherwise meaningful signal.

Anomaly Detection Techniques

Simple Statistical Methods

The simplest approach to identifying irregularities in data is to flag the data points that deviate from common statistical properties of a distribution, including mean, median, mode, and quantiles. Let's say the definition of an anomalous data point is one that deviates by a certain standard deviation from the mean. Traversing mean over time-series data isn't exactly trivial, as it's not static. You would need a rolling window to compute the average across the data points. Technically, this is called a rolling average or a moving average, and it's intended to smooth short-term fluctuations and highlight long-term ones. Mathematically, an n-period simple moving average can also be defined as a "low pass filter." (A Kalman filter is a more sophisticated version of this metric; you can find a very intuitive explanation of it here.)

Challenges

The low pass filter allows you to identify anomalies in simple use cases, but there are certain situations where this technique won't work. Here are a few:

- The data contains noise which might be similar to abnormal behavior, because the boundary between normal and abnormal behavior is often not precise.
- The definition of abnormal or normal may frequently change, as malicious adversaries constantly adapt themselves. Therefore, the threshold based on moving average may not always apply.
- The pattern is based on seasonality. This involves more sophisticated methods, such as decomposing the data into multiple trends in order to identify the change in seasonality.

Machine Learning-Based Approaches

Below is a brief overview of popular machine learning-based techniques for anomaly detection.

Density-Based Anomaly Detection

Density-based anomaly detection is based on the k-nearest neighbors algorithm.

Assumption: Normal data points occur around a dense neighborhood and abnormalities are far away.

The nearest set of data points are evaluated using a score, which could be Eucledian distance or a similar measure dependent on the type of the data (categorical or numerical). They could be broadly classified into two algorithms:

1. K-nearest neighbor: k-NN is a simple, non-parametric lazy learning technique used to classify data based on similarities in distance metrics such as Eucledian, Manhattan, Minkowski, or Hamming distance.
2. Relative density of data: This is better known as local outlier factor (LOF). This concept is based on a distance metric called reachability distance.

Clustering-Based Anomaly Detection

Clustering is one of the most popular concepts in the domain of unsupervised learning.

Assumption: Data points that are similar tend to belong to similar groups or clusters, as determined by their distance from local centroids.

K-means is a widely used clustering algorithm. It creates 'k' similar clusters of data points. Data instances that fall outside of these groups could potentially be marked as anomalies.

Support Vector Machine-Based Anomaly Detection

A support vector machine is another effective technique for detecting anomalies. A SVM is typically associated with supervised learning, but there are extensions (OneClassCVM, for instance) that can be used to identify anomalies as an unsupervised problems (in which training data are not labeled). The algorithm learns a soft boundary in order to cluster the normal data instances using the training set, and then, using the testing instance, it tunes itself to identify the abnormalities that fall outside the learned region.

Depending on the use case, the output of an anomaly detector could be numeric scalar values for filtering on domain-specific thresholds or textual labels (such as binary/multi labels).

Building a Simple Detection Solution Using a Low-Pass Filter

In this section, we will focus on building a simple anomaly-detection package using moving average to identify anomalies in the number of sunspots per month in a sample dataset, which can be downloaded here using the following command:

```
wget -c -b http://www-personal.umich.edu/~mejn/cp/data/sunspots.txt
```

The file has 3,143 rows, which contain information about sunspots collected between the years 1749-1984. Sunspots are defined as dark spots on the surface of the sun. The study of sunspots helps scientists understand the sun's properties over a period of time; in particular, its magnetic properties.

Moving Average Using Discrete Linear Convolution

Convolution is a mathematical operation that is performed on two functions to produce a third function. Mathematically, it could be described as the integral of the product of two functions, after one is reversed and shifted: $f * g(t) = \int_{-\infty}^{\infty} f(T) * g(t - T)dT$, where *f(T)* is an input function containing the quantity of interest (e.g. sunspot count at time *T*). *g(t —T)* is the weighting function shifted by an amount *t*. This way as *t* changes, different weights are assigned to the input function *f(T)*. In our case, *f(T)* represents the sunspot counts at time T. *g(t —T)* is the moving average kernel.

```
from __future__ import division
from itertools import izip, 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 and upload the same to dataset directory
#    Load the sunspot dataset as an Array
!mkdir -p dataset
!wget -c -b http://www-personal.umich.edu/~mejn/cp/data/sunspots.txt -P dataset
data = loadtxt("dataset/sunspots.txt", float)

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

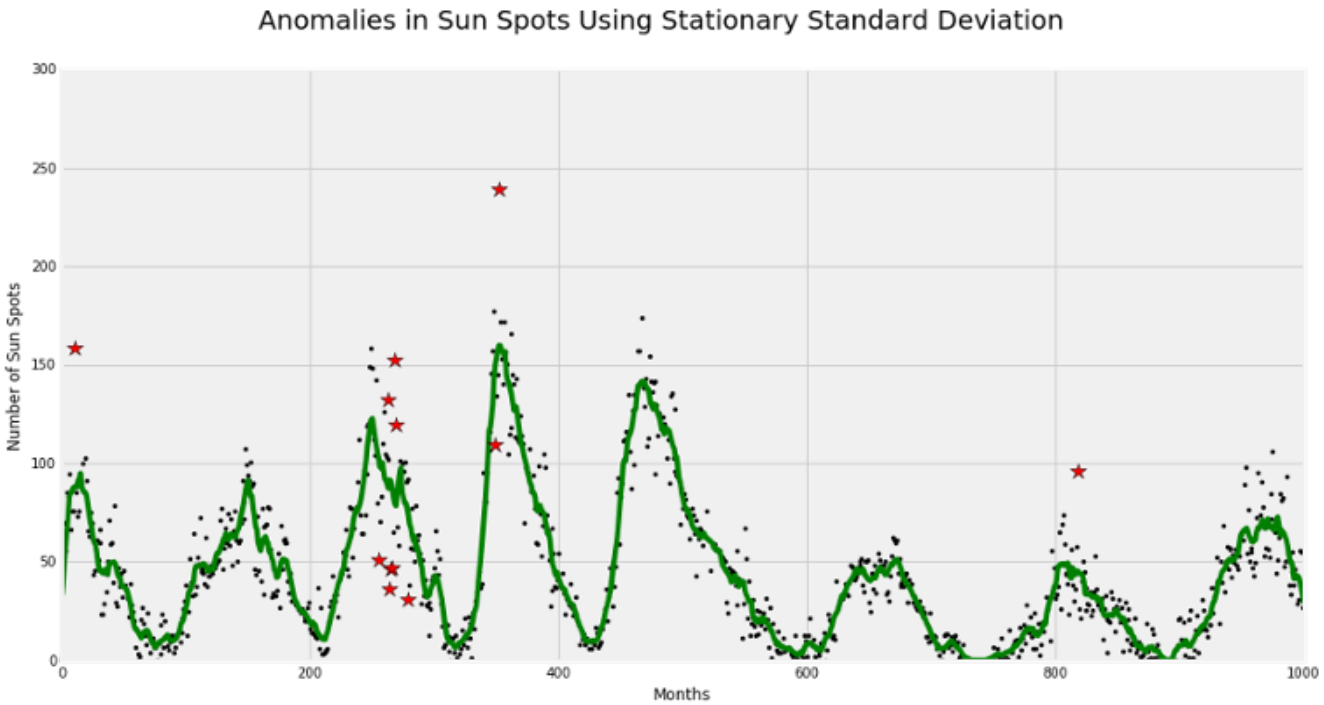
	Months	SunSpots
0	0.0	58.0
1	1.0	62.6
2	2.0	70.0
3	3.0	55.7
4	4.0	85.0

```
# 3. Lets define some use-case specific UDF(User Defined Functions)

def moving_average(data, window_size):
    """ Computes moving average using discrete linear convolution of two one dimensional
sequences.
    Args:
    -----
            data (pandas.Series): independent variable
            window_size (int): rolling window size

    Returns:
    --------
            ndarray of linear convolution

    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)
    return np.convolve(data, window, 'same')


def explain_anomalies(y, window_size, sigma=1.0):
  """ Helps in exploring the anamolies using stationary standard deviation
    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).tolist()
    residual = y - avg
    # Calculate the variation in the distribution of the residual
    std = np.std(residual)
    return {'standard_deviation': round(std, 3),
            'anomalies_dict': collections.OrderedDict([(index, y_i) for
                                                       index, y_i, avg_i in izip(count(),
y, avg)
              if (y_i > avg_i + (sigma*std)) | (y_i < avg_i - (sigma*std))])}


def explain_anomalies_rolling_std(y, window_size, sigma=1.0):
    """ Helps in exploring the anamolies using rolling standard deviation
    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_as_df = pd.DataFrame(testing_std)
    rolling_std = testing_std_as_df.replace(np.nan,
                                    testing_std_as_df.ix[window_size -
1]).round(3).iloc[:,0].tolist()
```

```
    std = np.std(residual)
    return {'stationary standard_deviation': round(std, 3),
            'anomalies_dict': collections.OrderedDict([(index, y_i)
                                                       for index, y_i, avg_i, rs_i in
izip(count(),

  y, avg_list, rolling_std)
                if (y_i > avg_i + (sigma * rs_i)) | (y_i < avg_i - (sigma * rs_i))])}


# This function is repsonsible for displaying how the function performs on the given
dataset.
def plot_results(x, y, window_size, sigma_value=1,
                 text_xlabel="X Axis", text_ylabel="Y Axis", applying_rolling_std=False):
    """ Helps in generating the plot and flagging the anamolies.
        Supports both moving and stationary standard deviation. Use the
'applying_rolling_std' to switch
        between the two.
    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
    """
    plt.figure(figsize=(15, 8))
    plt.plot(x, y, "k.")
    y_av = moving_average(y, window_size)
    plt.plot(x, y_av, color='green')
    plt.xlim(0, 1000)
    plt.xlabel(text_xlabel)
    plt.ylabel(text_ylabel)

    # Query for the anomalies and plot the same
    events = {}
    if applying_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'].iterkeys(), dtype=int,
count=len(events['anomalies_dict']))
    y_anomaly = np.fromiter(events['anomalies_dict'].itervalues(), 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)
```

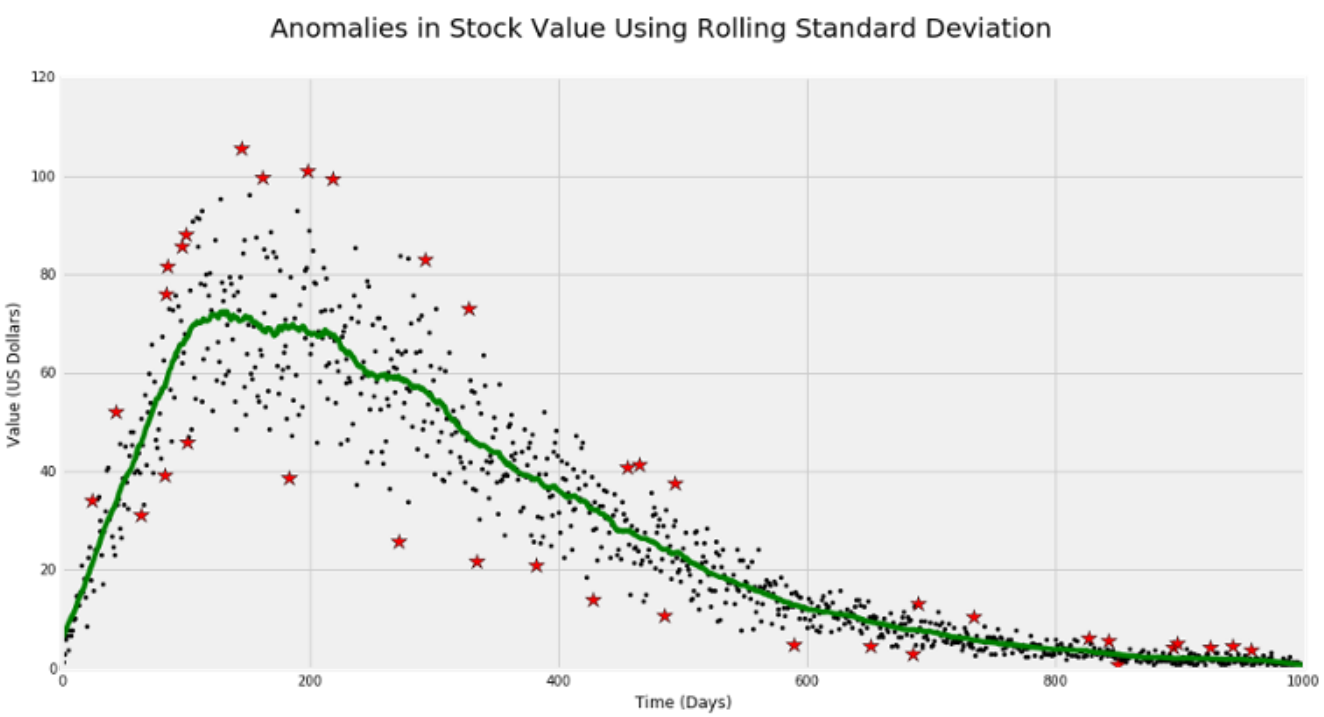
```
    plt.show()
# 4. Lets play with the functions
x = data_as_frame['Months']
Y = data_as_frame['SunSpots']

# plot the results
plot_results(x, y=Y, window_size=10, text_xlabel="Months", sigma_value=3,
             text_ylabel="No. of Sun spots")
events = explain_anomalies(y, window_size=5, sigma=3)

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

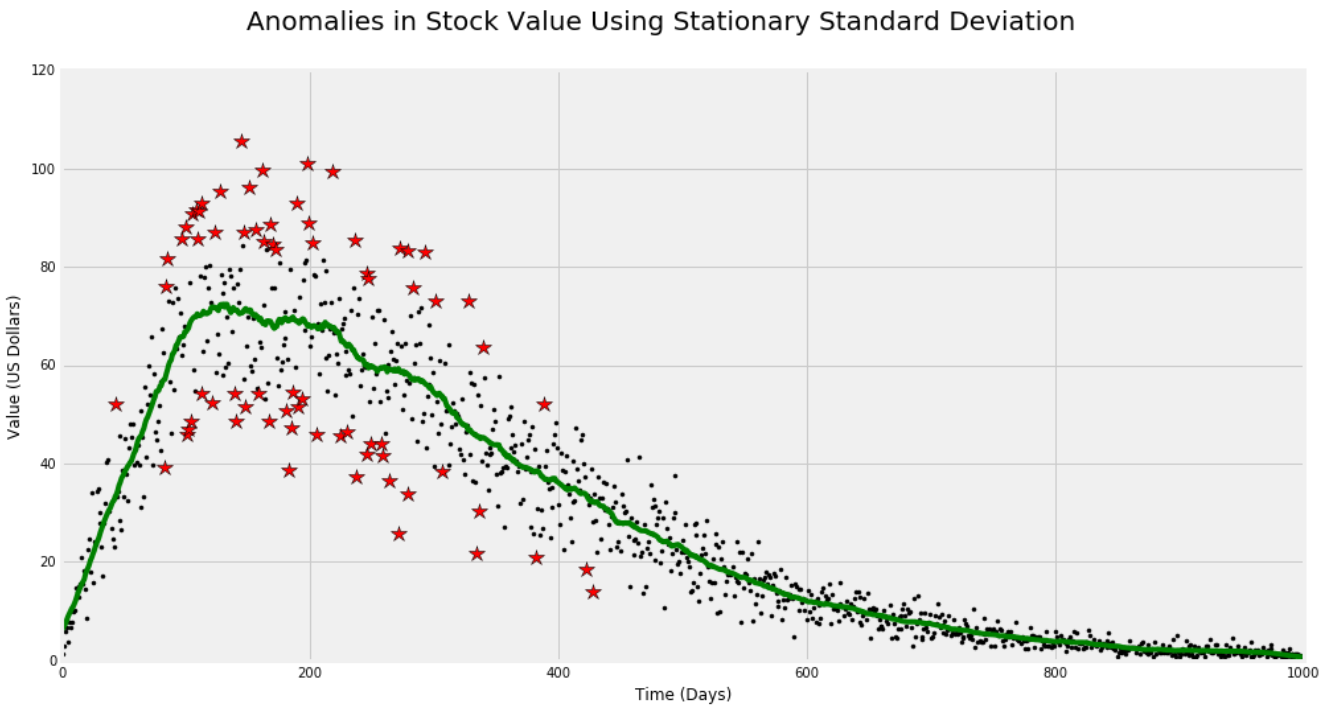


Let's see if the above anomaly detection function could be used for another use case. Let's assume that we generate a random dataset that hypothetically relates to Company A's stock value over a period of time. The x axis represents time in days (since 2013) and the y axis represents the value of the stock in dollars.

```
# Convenience function to add noise
def noise(yval):
    """ Helper function to generate random points """
    np.random.seed(0)
    return 0.2*np.asarray(yval)*np.random.normal(size=len(yval))

# Generate a random dataset
def generate_random_dataset(size_of_array=1000, random_state=0):
    """ Helps in generating a random dataset which has a normal distribution
    Args:
    -----
        size_of_array (int): number of data points
        random_state (int): to initialize a random state

    Returns:
```

```
    --------
        a list of data points for dependent variable, pandas.Series of independent
variable
    """
    np.random.seed(random_state)
    y = np.random.normal(0, 0.5, size_of_array)
    x = range(0, size_of_array)
    y_new = [y_i + index**((size_of_array - index)/size_of_array) +  noise()
     for index, y_i in izip(count(), y)]
    return x, pd.Series(y_new)


# Lets play
x1, y1 = generate_random_dataset()
# Using stationary standard deviation over a continuous sample replicating
plot_results(x1, y1, window_size=12, title_for_plot="Statinoary Standard Deviation",
                    sigma_value=2, text_xlabel="Time in Days", text_ylabel="Value in $")

# using rolling standard deviation for
x1, y1 = generate_random_dataset()
plot_results(x1, y1, window_size=50, title_for_plot="Using rolling standard deviation",
             sigma_value=2, text_xlabel="Time in Days", text_ylabel="Value in $",
applying_rolling_std=True)
```





Looks like our anomaly detector is doing a decent job. It is able to detect data points that are 2 sigma away from the fitted curve. Depending on the distribution of a use case in a time-series setting, and the dynamicity of the environment, you may need to use stationary (global) or non-stationary (local) standard deviation to stabilize a model. The mathematical function around the standard deviation could be modified very easily to use a customized formulation.

Note: The analyses above are intended is to highlight how you can quickly build a simple anomaly detector. They are not necessarily the most efficient solution.

Conclusion

Now, you have some introductory knowledge of anomaly detection, including how to use low-pass filter and simple moving average to detect abnormalities. In an upcoming tutorial, we will dive deeper into techniques that address more specific use cases in the most efficient way possible.