Anomaly detection

[Anomaly detection](https://en.wikipedia.org/wiki/Anomaly_detection) (or outlier detection) is the identification of rare items, events or observations which raise suspicions by differing significantly from most of the data. Typically, anomalous data can be connected to some kind of problem or rare event such as e.g. bank fraud, medical problems, structural defects, malfunctioning equipment etc. This connection makes it very interesting to be able to pick out which data points can be considered anomalies, as identifying these events are typically very interesting from a business perspective.

This brings us to one of the key objectives: How do we identify whether data points are normal or anomalous? In some simple cases, as in the example figure below, [data visualization](https://en.wikipedia.org/wiki/Data_visualization) can give us important information.

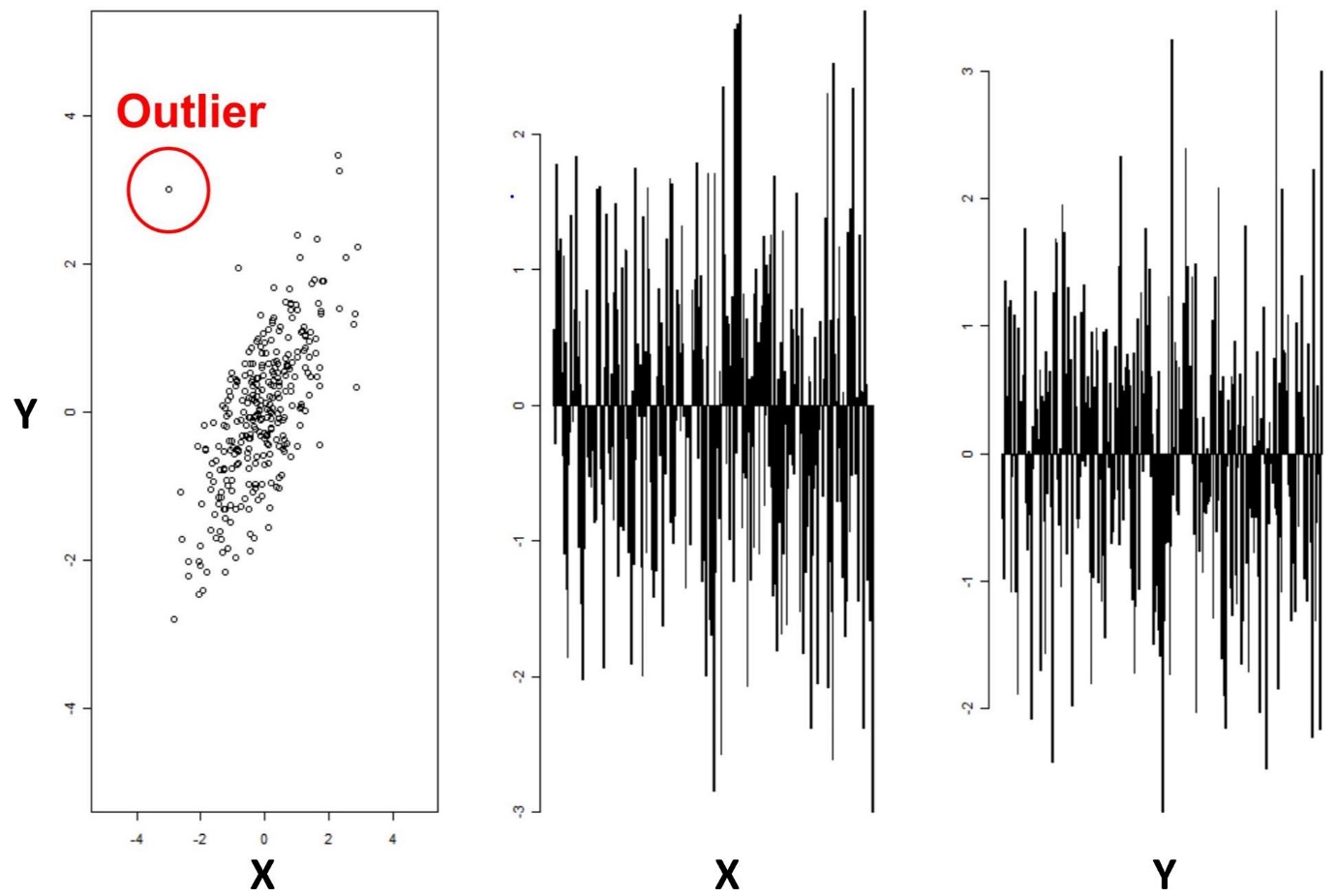


Figure 1 : Anomaly detection for two variables

In this case of two-dimensional data (X and Y), it becomes quite easy to visually identify anomalies through data points located outside the typical [distribution](https://en.wikipedia.org/wiki/Probability_distribution). However, looking at the figures to the right, it is not possible to identify the outlier directly from investigating one variable at the time: It is the combination of the X and Y variable that allows us to easily identify the anomaly. This complicates the matter substantially when we scale up from two variables to 10–100s of variables, which is often the case in practical applications of anomaly detection.

**Data Pre-processing and Class Imbalance:**

If it is the case that one of the features is considered redundant, we should be able to summarize the data with less characteristics (features). So, the way PCA tackles this problem is: Instead of simply picking out the useful features and discarding the others, it uses a linear combination of the existing features and constructs some new features that are good alternative representation of the original data. In our 2D toy dataset, PCA will try to pick the best single direction, or often referred to as **first principal components** in 2D, and project our points onto that single direction. So the next question becomes, out of the many possible lines in 2D, what line should we pick?

It turns out, there are two different answers to this question. First answer is that we are looking for some features that strongly differ across data points, thus, PCA looks for features that captures as much variation across data points as possible. The second answer is that we are looking for the features that would allow us to "reconstruct" the original features. Imagine that we come up with a feature that has no relation to the original ones; If we were to use this new feature, there is no way we can relate this to the original ones. So PCA looks for features that minimizes the reconstruction error. These two notions can be depicted in the graph below, where the black dots represents the original data point, the black line represents the projected line, the red dot on the left shows the points on the projected line and the red line on the right shows the reconstruction error.

This node performs a [principal component analysis (PCA)](http://en.wikipedia.org/wiki/Principal_component_analysis) on the given data using the Apache Spark [implementation](http://spark.apache.org/docs/latest/ml-features.html#pca). The input data is projected from its original feature space into a space of (possibly) lower dimension with a minimum of information loss.

## Options

**Fail if missing values are encountered**

If checked, execution fails, when the selected columns contain missing values. By default, rows containing missing values are ignored and not considered in the computation of the principal components.

**Target dimensions**

Select the number of dimensions the input data is projected to. You can select either one of:

* Dimensions to reduce to: Directly specify the number of target dimensions. The specified number must be lower or equal than the number of input columns.
* Minimum information fraction to preserve (%): Specify the fraction in percentage of information to preserve from the input columns. This option requires Apache Spark 2.0 or higher.

**Replace original data columns**

If checked, the projected DataFrame/RDD will not contain columns that were included in the principal component analysis. Only the projected columns and the input columns that were not included in the principal component analysis remain.

**Columns**

Select columns that are included in the analysis of principal components, i.e the original features.

SMOTE is an oversampling algorithm that relies on the concept of nearest neighbours to create its synthetic data. Proposed back in [2002 by Chawla et. al](https://arxiv.org/abs/1106.1813)., SMOTE has become one of the most popular algorithms for oversampling.

The simplest case of oversampling is simply called oversampling or up sampling, meaning a method used to duplicate randomly selected data observations from the outnumbered class.

Oversampling’s purpose is for us to feel confident the data we generate are real examples of already existing data. This inherently comes with the issue of creating more of the same data we currently have, without adding any diversity to our dataset, and producing effects such as overfitting.

Hence, if overfitting affects our training due to randomly generated, up sampled data– or if plain oversampling is not suitable for the task at hand– we could resort to another, smarter oversampling technique known as synthetic data generation.

Synthetic data is intelligently generated artificial data that resembles the shape or values of the data it is intended to enhance. Instead of merely making new examples by copying the data we already have (as explained in the last paragraph), a synthetic data generator creates data that is like the existing one. Creating synthetic data is where SMOTE shines.

Example of an imbalanced dataset

For each observation that belongs to the under-represented class, the algorithm gets its K-nearest-neighbors and synthesizes a new instance of the minority label at a random location in the line between the current observation and its nearest neighbor.

In our example (shown in the next image), the blue encircled dot is the current observation, the blue non-encircled dot is its nearest neighbor, and the green dot is the synthetic one.