**Basics of Anomaly Detection with Multivariate Gaussian Distribution**

Overview of anomaly detection, review of multivariate Gaussian distribution, and implementation of basic anomaly detection algorithm in Python with two examples

[[](https://viyaleta.medium.com/?source=post_page-----65aff59949b7--------------------------------)](https://viyaleta.medium.com/?source=post_page-----65aff59949b7--------------------------------)

[[Towards Data Science](https://towardsdatascience.com/?source=post_page-----65aff59949b7--------------------------------)](https://towardsdatascience.com/?source=post_page-----65aff59949b7--------------------------------)

[Viyaleta Apgar](https://viyaleta.medium.com/?source=post_page-----65aff59949b7--------------------------------)

·

Follow

Published in

[Towards Data Science](https://towardsdatascience.com/?source=post_page-----65aff59949b7--------------------------------)

·

11 min read

·

Jul 5

415

3

Our innate ability to recognize patters allows us to use this skill in filling-in gaps or predicting what is going to happen next. Occasionally, however, something happens that does not fit our expectation and does not fall into our perception of a pattern. We call such occurrences anomalies. If we are trying to predict something, we may want to exclude anomalies from our training data. Or perhaps we want to identify anomalies to help make our life better. In either case, anomaly detection techniques can prove to be useful and applicable in most industries and subject areas.

This article will guide you through the basics of anomaly detection and implementation of statistical anomaly detection model.

**What is anomaly detection?**

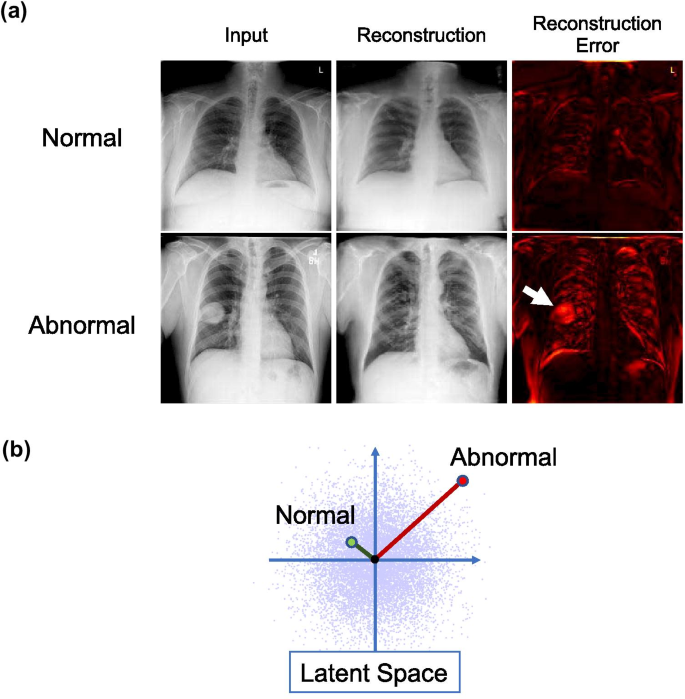
In general terms, anomaly detection refers to the process of identifying phenomena that is out of ordinary. The goal of anomaly detection is to identify events, occurrences, data points, or outcomes that are not in line with our expectations and do not fit some underlying pattern. Hence, the key to implementing anomaly detection is to understand the underlying pattern of expected events. If we know the pattern of the expected, we can use it to map the never-before-seen data points; if our mapping is not successful and our new data point falls outside of our expected pattern, it’s probable that we have found our anomaly.

There are three types of anomalies that typically occur. First type includes individual instances which are considered anomalous with respect to the entire dataset (e.g., an individual car driving at very low speed on a highway is anomalous compared to all highway traffic). Second type includes instances which are anomalies within a specific context (e.g., credit card transactions which appear OK when compared to all credit card transactions but are anomalous for the specific individual’s spending pattern). Third type of anomalies is collective — a set of instances may be considered anomalous even though each instance on its own follows a certain expectation (e.g., a single fraudulent credit card transaction on Amazon may not seem out of ordinary but a set of transactions that take place back to back in a short amount of time is suspicious) [1].

Anomaly detection techniques fall into three categories:

1. **Supervised detection**requires positive and anomalous labels in the dataset. Supervised learning algorithms like neural networks or boosted forests can be applied to categorize data points into expected/anomaly classes. Unfortunately, anomaly datasets tend to be very imbalanced and generally do not have enough training samples to allow up or downsampling techniques to aid the supervised learning.
2. **Semi-supervised detection** deals with data that is partially labeled. Semi-supervised techniques assume that the input data only contains positive instances and that the input data follows an expected pattern. These techniques attempt to learn the distribution of positive cases in order to be able to generate positive instances. During testing, the algorithm will evaluate the likelihood that the anomalous instance could have been generated by the model and uses this probability to predict anomalous cases. [2]
3. **Unsupervised detection**uses completely unlabeled data in order to create a boundary of expectation and anything that falls outside of this boundary is considered to be anomalous.

Anomaly detection techniques can be applied to any data and data format impacts which algorithm will be most useful. Types of data include **series** (time series, linked list, language, sound), **tabular** (e.g., engine sensor data), **image** (e.g., X-ray images), and **graph** (e.g., workflow or process).



Example of anomaly detection in chest X-ray images[[4]](https://doi.org/10.1007/s10278-020-00413-2)

Given the variety of problems and techniques, anomaly detection is actually a vast area of data science with many applications. Some of these applications include: fraud detection, cybersecurity applications, analysis of sales or transactional data, identification of rare diseases, monitoring of manufacturing processes, exoplanet search, machine learning preprocessing, and many more. Therefore, access to powerful and performant algorithms has the potential to make significant impact in many fields.

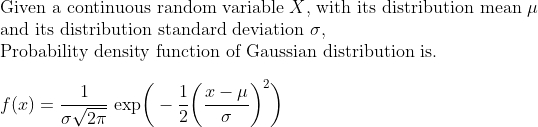
Let’s take a look how at the most basic algorithm that can be used to detect anomalies.

**Gaussian Distribution for Anomaly Detection**

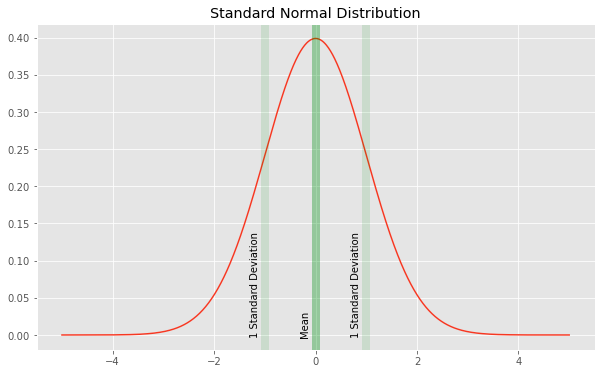
One of the basic anomaly detection techniques employs the power of Gaussian (i.e. Normal) distribution in order to identify outliers.

Discovered by Carl Friedrich Gauss, Gaussian distribution models many natural phenomena and is, therefore, a popular choice for modeling features in a dataset. This distribution’s probability density function is a bell curve centered at the arithmetic mean and the width of the curve is defined by the variance of the dataset. With the majority of the cases being at or near the center, the probability density function features two elongated tails on each end. The more rare the instance — the further it is from the center — the more likely it is to be an outlier or an anomaly. Eureka!— we can use this concept to model anomalies in our dataset.

The probability density function, defined as f(x), measures the probability of some outcome x in our dataset. Formally,



Let’s assume that our dataset had only one feature and that feature followed a normal distribution, then we can model our anomaly detection algorithm using f(x) from above. We can then set some threshold epsilon which will determine if a case is anomalous or not. Epsilon should be set heuristically and its value will depend on the use case and the preferred sensitivity for anomalies.

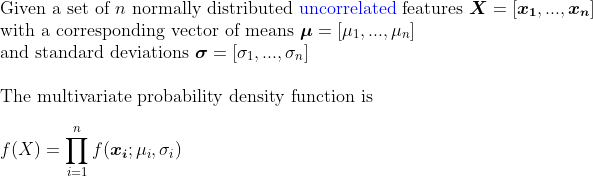


Graph depicting Normal Distribution [[5]](https://github.com/viyaleta/Anomaly-Detection/blob/main/Examples/1%20Anomaly%20Detection%20with%20Guassian%20Distribution.ipynb)

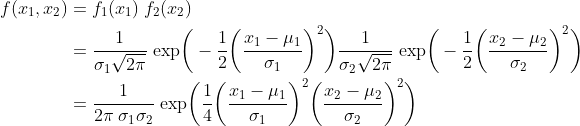
In a normal distribution, 2.5% of instances occur two standard deviations below the mean value. So if we set our threshold to 0.054, then about 2.5% of events in our dataset will be classified as anomalies (CDF of 2 standard deviations below the mean is 2.5 and PDF at -2 is 0.054). Lower thresholds will yield fewer classified anomalies and higher thresholds will be less sensitive.

In real world, there is likely to be a tradeoff as some of positive cases may fall below the threshold and some of the anomalies may hide above the threshold. It will be necessary to understand the use case and test different epsilon values before settling on the one that is best suited.

An example with a single feature is trivial — what do we do if we have more than one feature? If our features are completely independent, we can actually take the product of the feature probability density function in order to classify anomalies.



For a two uncorrelated feature case, this becomes



Essentially, the product of probabilities of features can ensure that if at least one feature has an outlier, we can detect an anomaly (given that our epsilon is high enough); if our instance exhibits an outlier value in several features, our probability will be even smaller (since our total probability value is a product of fractions) and a value is even more likely to be an anomaly.

However, *we cannot assume that our features are independent*. And this is where a multivariate probability density function comes it. In the multivariate case, we build a covariance matrix (denoted by a Σ) in order to capture how the features are related to each other. Then, we can use the covariance matrix to avoid “double-counting” of feature relations (this is a very rudimentary way of phrasing what is actually happening). The formula for multivariate distribution probability density function is shown below and [these slides from Duke](https://www2.stat.duke.edu/courses/Spring12/sta104.1/Lectures/Lec22.pdf) do a good job and deriving the formula.

f of x one through x n equals to ratio between negative transpose of x minus mu times inverse of covariance matrix times difference of x and mu and square root of product of 2 pi to the nth power times determinant of covariance matrix

Here, x is an input vector, μ is a vector of feature means and Σ is a covariance matrix between the features.

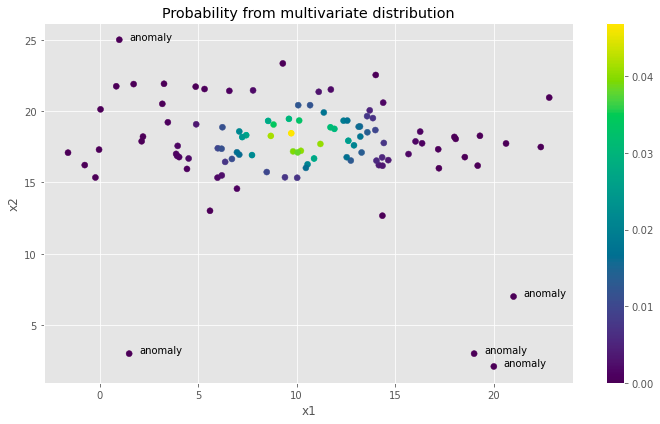
To make our life easier, we can use scipy library to implement this function: scipy.stats.multivariate\_normal takes as input a vector of feature means and standard deviations and has a .pdf method for returning probability density given a set of points.

Let’s try this implementation on an actual example.

**Two-feature Model Implementation in Python**

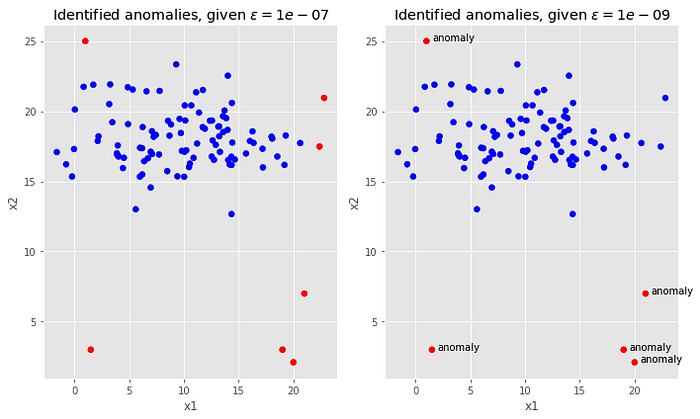
First, let’s observe a two-feature example which will allow us to visualize anomalies in Eucledian space. For this example, I generated two features with 100 samples drawn from the Normal distribution (these are the positive samples). I calculated feature means and standard deviations and fit a multivariate normal model from the scipy.stats library with the distribution information. **Of note:**I fit my model with positive samples only. In real-world data, we want to clean our dataset to ensure that the features follow normal distribution and do not contain outliers or odd values — this will improve models ability to locate anomalies (especially since it will help ensure the feature Normal distribution requirement). Finally, I added 5 anomalous samples to my dataset and use the .pdf method to report the probabilities.

The following scatterplot shows the result: x1 feature is plotted on the x-axis, x2 feature is plotted on the y-axis, anomalies are annotated, and the color represents the probability from the multivariate probability density function.



Scatterplot showing positive and anomalous points [[5]](https://github.com/viyaleta/Anomaly-Detection/blob/main/Examples/1%20Anomaly%20Detection%20with%20Guassian%20Distribution.ipynb)

Once we set our threshold *low enough*, we will be able to distinguish the anomalies from the expected values. Two charts below compare epsilon values between 1x10^-7 and 1x10^-9. Epsilon value of 1x10^-9 tends to capture our intended outliers better while 1x10^-7 identifies some positive samples as outliers.



Scatterplot comparison of identified anomalies at higher and lower epsilon [[5]](https://github.com/viyaleta/Anomaly-Detection/blob/main/Examples/1%20Anomaly%20Detection%20with%20Guassian%20Distribution.ipynb)

In this example, it is easy to identify the epsilon because we can visually depict and identify anomalies and analyze our results. Let’s see how this changes in an example with a few more features.

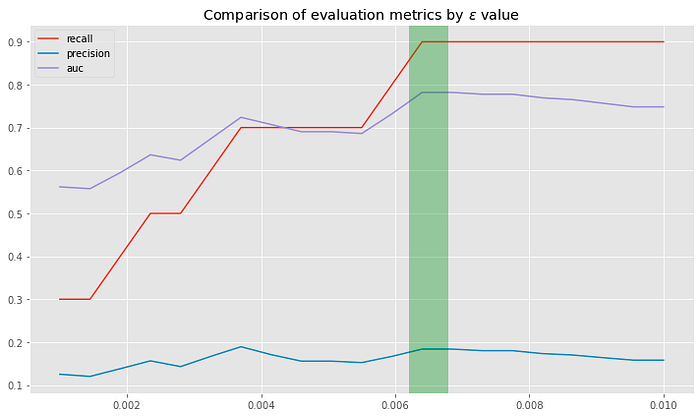
**Multivariate Model Implementation with Python**

For this example, I will use [the wine dataset](http://odds.cs.stonybrook.edu/wine-dataset/) from ODDS library [3]. This dataset contains 13 numerical features and 129 instances. The features capture information about the wine and the original dataset was used for classification tasks based on wine analysis. For the purpose of anomaly detection, one of the target classes was downsampled and is presented as an outlier. There are total of 10 anomalies among 129 instances (~8%). We are working with a fairly clean dataset with no missing values.

The very first thing we must do is ensure that our features follow a Gaussian distribution. Where possible, we should remove outliers and normalize the distribution using one of normalization tactics. In this dataset, 4 features already follow a normal distribution (alcohol, ash, alcalinity of ash, and non-flavanoid phenols) and 4 features can be normalized by taking their log (total phenols, proanthocyanins, color intensity, and hue). While better strategies exist for the remaining features, for the purpose of this exercise I simply dropped them from our training dataset. Finally, I removed the outliers by excluding all rows that contain at least one feature value that is above or below 2 standard deviations from the mean. The remainder of the code is the same as in the example above.

Unlike the two-feature example from the section above, it is no longer feasible to visualize the results on a 2-dimensional plane but we can use confusion matrix metrics (including recall and precision) and ROC area under the curve to help us find the correct epsilon for the use-case.

As there is usually a tradeoff between precision and recall, the setting of epsilon depends on the sensitivity requirement of our use case. For this example, I looked for an epsilon that maximizes the area under the curve. Some use cases may call for trying to find as many anomalies as possible (at the cost of including positive values) while other use-cases may call for only detecting anomalies if we are absolutely sure (at the cost of missing some anomalies from our final report). I calculated evaluation metrics for several different epsilon values.



Lineplot comparison of evaluation metrics by epsilon value [[5]](https://github.com/viyaleta/Anomaly-Detection/blob/main/Examples/1%20Anomaly%20Detection%20with%20Guassian%20Distribution.ipynb)

As epsilon increases, recall increases. Precision is fairly low throughout the proposed epsilon values but tends to peak around 0.0035 and 0.0065. The AUC attempts to strike the balance between precision and recall and has a peak around 0.0065. Let’s take a look at the confusion matrix.

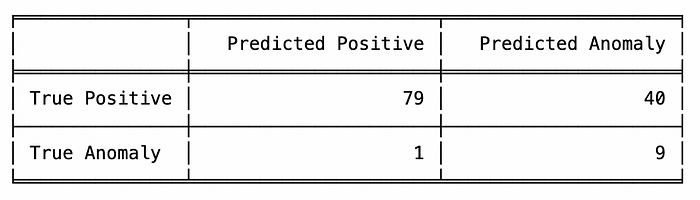


Table depicting confusion matrix [[5]](https://github.com/viyaleta/Anomaly-Detection/blob/main/Examples/1%20Anomaly%20Detection%20with%20Guassian%20Distribution.ipynb)

Our model does quite well at finding all of the anomalies and only misses one. This is a fantastic result given that I excluded a third of the features. Unfortunately, our model also shows 40 positive instances as anomalies which means that if we use the model for anomaly detection, we would have to manually check half of positive instances to see if they were actually anomalous.

To improve this model we can further engineer the remaining features and find an epsilon values that may be a bit less sensitive to outliers. The rest of this problem is trivial and is left as an exercise for the reader (iykyk). You can find the [source code here](https://github.com/viyaleta/Anomaly-Detection/blob/main/Examples/1%20Anomaly%20Detection%20with%20Guassian%20Distribution.ipynb).

**Potential Drawbacks with Gaussian Anomaly Detection**

Multivariate Gaussian distribution is a great model for anomaly detection — it is simple, fast, and easy to execute. However, its drawbacks can prevent its utilization for numerous use cases.

**First**, multivariate distribution can produce fairly low probability density values. Generally, this is not a problem for modern computers. But there may be instances where the values are too low to be effectively handled by a computer.

**Second**, we must ensure that our features follow a normal distribution. This may not be too much of an issue if time and effort is dedicated to perform proper feature engineering and data manipulation but putting effort is risky since we won’t know the payout until we do the work.

**Third**, this model does not handle categorical features and if our dataset includes categorical features, we must create a separate model per each combination of categorical features (which can turn out to be a lot of work).

**Finally**, the model assumes that all features are equally relevant and there is no complex relationship between the features. One option to deal with this is to implement the multivariate distribution probability density function from scratch and include some parameter to help with feature importance. To resolve the issue with feature relationships, we could do further feature engineering and create new features but this process can be difficult, time-consuming, and risky (in terms of payout).

Nonetheless, the use of multivariate Gaussian distribution for anomaly detection is a great first step for tabular anomaly detection problems. It can be used to set a benchmark or can prove to be a perfect tool for catching anomalies in a dataset and provides for us an intuitive way to understand anomaly detection.

Thanks for reading! In the near future I hope to do a full series on anomaly detection so if this topic is interesting to you, stay tuned.

**Sources:**

1. <https://www.kaggle.com/code/matheusfacure/semi-supervised-anomaly-detection-survey>
2. <https://ai.googleblog.com/2023/02/unsupervised-and-semi-supervised.html>
3. Saket Sathe and Charu C. Aggarwal. [LODES: Local Density meets Spectral Outlier Detection.](http://saketsathe.net/papers/lodes.pdf) SIAM Conference on Data Mining, 2016.
4. Nakao, T., Hanaoka, S., Nomura, Y. *et al.* Unsupervised Deep Anomaly Detection in Chest Radiographs. *J Digit Imaging* **34**, 418–427 (2021). <https://doi.org/10.1007/s10278-020-00413-2>
5. <https://github.com/viyaleta/Anomaly-Detection/blob/main/Examples/1%20Anomaly%20Detection%20with%20Guassian%20Distribution.ipynb>

Math typesetting courtesy of [Codecogs online LaTeX editor](https://latex.codecogs.com/eqneditor/editor.php).

Jupyter Notebook with examples [can be found here](https://github.com/viyaleta/Anomaly-Detection/blob/main/Examples/1%20Anomaly%20Detection%20with%20Guassian%20Distribution.ipynb).