Anomaly Detection

Introduction

Anomaly Detection is a method for detecting outliers in numerical data. It's considered a semi-supervised learning technique, since it doesn't require labelled data (each datum defined as anomalous or not) to train the model, but does require it to set hyper-parameters i.e. a cut-off point for defining whether a data point is an anomaly or not.

It's often used in manufacturing to detect if a unit meets quality requirements or if systems (e.g. fuel pumps, CPUs, motors) are operating normally, among other applications.

It's especially useful where there are many non-anomalous data and only a few anomalous examples. In this situation, models such as logistic regression or a neural network have little data to learn how to predict anomalous examples, and will appear to perform well by just classifying all examples as non-anomalous.

Problem Statement

Let's say we are applying anomaly detection to in-service aircraft to detect if a wing-tank pump has failed. We have data from a test aircraft, where the pump failed twice and the failure was detected and labelled thanks to the extra sensors onboard.

There are two sensors related to the pump that are actually shared between in-service and test aircraft, so these will be used to detect pump failure (anomaly) as the additional sensors on the test aircraft wouldn't be available in situ.

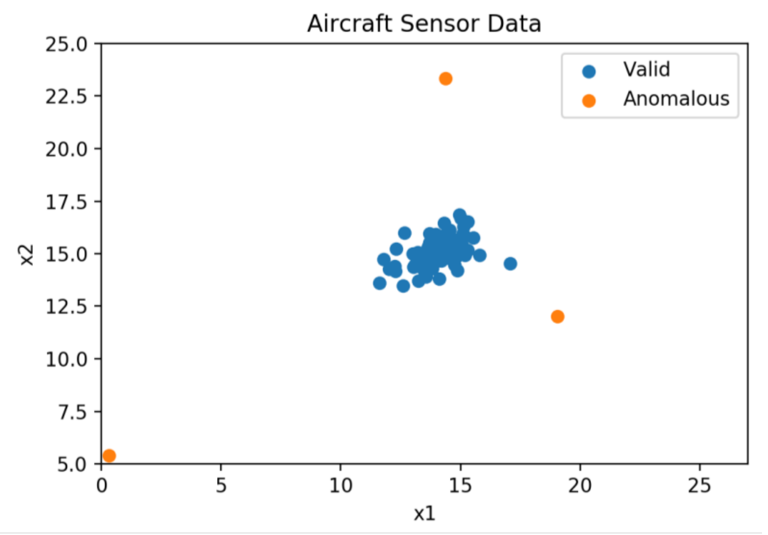

Figure 1 - Aircraft Sensor Data

Building the Model

To build the model, the data should first be separated into training, cross-validation and test sets in a 60/20/20 split (or more heavily weighted toward training data for large datasets). The training data will just be used to define the basic part of the model, so doesn't need to be labelled, however the cross-validation and test sets will be used for model tuning and performance analysis, so labelled data are required for these.

Next, the training data's Gaussian distribution is calculated (in one or more dimensions). This is a probability density function, meaning it returns *the probability of occurrence* (P), i.e. the probability a data point will be present at a given coordinate. P values can be defined at individual data-points in addition to the overall distribution.

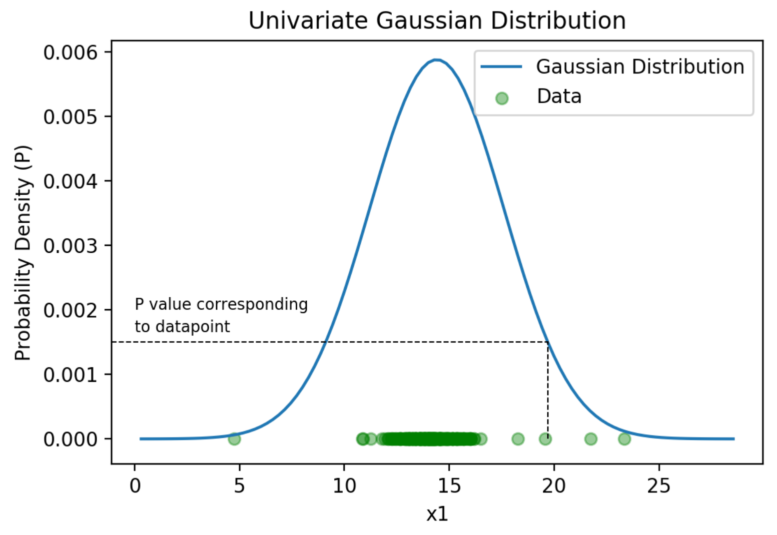

Figure 2 - Univariate Gaussian Distribution

The model classifies a data point as anomalous or not by computing its P value, and if it's below a cut-off point ε, it will be defined as anomalous. Setting this cut-off correctly is key to good model performance, and is often a trade-off between getting more false positives (incorrectly defined as anomalous) or more false negatives (incorrectly defined as normal), the correct balance of which is application dependent.

To find a good ε value, P values for the cross-validation dataset are computed. Since this dataset is labelled we can use it to test the performance of various ε values by classifying the data into normal/anomalous classes and comparing it to the correct labels. The optimal ε is that for which the most data are correctly classified.

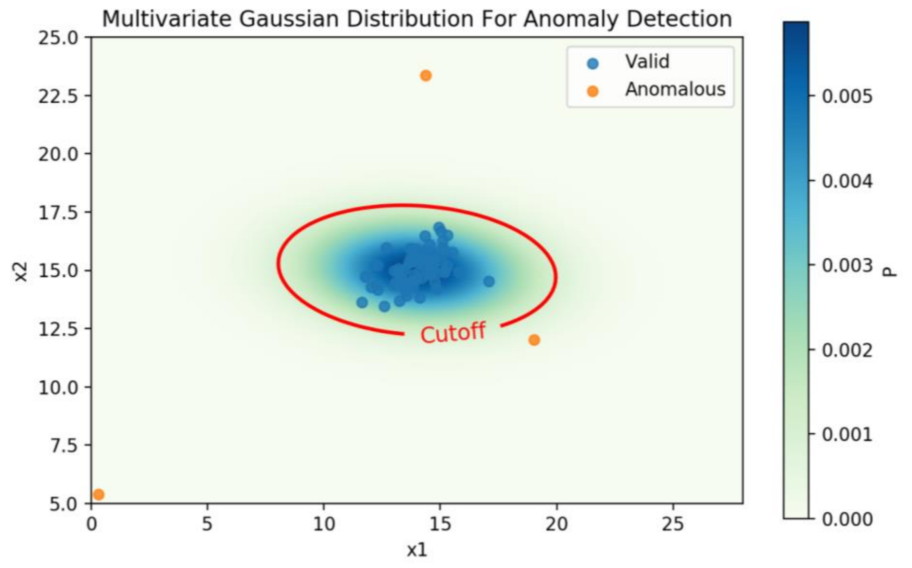

Figure 3 - Multivariate Gaussian distribution with P contour plot and cutoff

P values are found for new data and along with ε, can be used to define them as anomalous or not.

Implementing Anomaly Detection

Source code can be found at: P:\ENGINEERING\FPO\Digitalisation\1 FPO Projects\IIX Data Analytics\02-General presentations\Knowledge folder - preparation\Data Science Learning Framework\Machine Learning Posters\6 – Anomaly Detection

Introduction

To implement this model, we'll use the example above of identifying pump failures based on sensor data, however with 50 sensors as opposed to 2. This will make it harder to visualise the results, but shows how the model can be applied in highly dimensional data, and having much more data gives the model more fidelity.

	machine_status	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sensor_05	
0	0	2.465394	47.09201	53.2118	46.310760	634.3750	76.45975	
1	0	2.465394	47.09201	53.2118	46.310760	634.3750	76.45975	
2	0	2.444734	47.35243	53.2118	46.397570	638.8889	73.54598	…
3	0	2.460474	47.09201	53.1684	46.397568	628.1250	76.98898	
4	0	2.445718	47.13541	53.2118	46.397568	636.4583	76.58897	

Table 1 - Input Data

The data has 205,836 normal rows and 14,484 anomalous rows, this is strongly skewed with only 7% of the data labelled as positive. Other models may train poorly here and simply optimise by classifying all data as negative, but this is where anomaly detection shines.

The Train/Cross-Validation/Test Split

The first step is correctly splitting your data correctly. It's good practice to put only negative data in the training set, and splitting the positive examples evenly between the cross-validation and test sets (including negative examples in these sets too!). There are two reasons for this:

1. By only including negative examples in the training set, you ensure the Gaussian distribution only represents these data.
2. If you have few positive examples, it's best to save these for cross-validation and testing where they'll be more useful.

The Gaussian Distribution

Anomaly detection can use the univariate or multivariate Gaussian distribution, but we'll use the multivariate since it performs better in most applications.

Calculating the Gaussian distribution is the next step in anomaly detection, and requires 2 parameters from the training data: μ – the mean of each column (vector), and Σ – the covariance matrix, which can be computed using:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

Eq. 1

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

Eq. 2

m – number of training examples
$x^{(i)}$ - i^{th} training example

These can be used as constants to calculate the probability density P at a given vector x (or a given datum from the cross-validation/test set):

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}}\exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

Eq. 3

Optimising ε

Now we have our Gaussian distribution we can use input data x from the cross-validation set to return their respective $p(x)$. Before we use this to optimise ε, we need to understand the metric by which we'll optimise.

Precision and recall are two metrics for measuring a models success, their equations are:

$$\text{Precision} = \frac{\#\text{ of True Positives}}{\#\text{ of True Positives} + \#\text{ of False Positives}}$$

Eq. 4

$$\text{Recall} = \frac{\#\text{ of True Positives}}{\#\text{ of True Positives} + \#\text{ of False Negatives}}$$

Eq. 5

True Positive – Datum correctly identified as anomalous
True Negative – Datum correctly identified as normal
False Positive – Datum incorrectly identified as anomalous
False Negative – Datum incorrectly identified as normal

Maximising both of these (minimising incorrectly assigned data) is the key to good performance, so by creating maximizing the length of the vector they make, you can optimise ε:

$$\max(\sqrt{\text{precision}^2 + \text{recall}^2})$$

Eq. 6

Once P has been found for data in the cross-validation set, initialise many (~1000) logarithmically spaced instances of ε between 0 and the maximum P value. Iterate over these, for each defining data as anomalous if $p(x) < \varepsilon$, then calculate precision, recall and their corresponding vector. Pick value of ε with the longest vector and the model is complete.

Assessing Model Performance

Finally, we can assess the performance of our model on the test dataset. We must compute $p(x)$ for data within it, the define them as anomalous if $p(x) < \varepsilon$, then we can use a confusion matrix for analysis.

A confusion matrix is a way of displaying the true/false positives/negatives, sorting by the actual anomalous/normal categories column-wise, and the predicted anomalous/normal categories row-wise. Each entry will be the number of data in the test set that fall into that category:

		Actual	
		Anomalou	Normal
Predicted	Anomalous	TP	FP
	Normal	FN	TN

Correctly assigned (want to maximise)

Incorrectly assigned (want to minimise)

Table 2

For our model:

		Actual	
		Anomalou	Normal
Predicted	Anomalous	1479	334
	Normal	6	4666

Table 3

So our model performed well, with 95% of data correctly assigned. It seems to be classifying many more false positives than false negatives, so incrementally raising ε small amounts may improve performance further.