**Coursework 2: Anomaly Detection**

Team Abyss:Zhongqi Zhuang, Elton Lam, John Huang and Nicholas Gerard

*Department of Chemical Engineering, Imperial College London, U.K.*

**1. Intuition**

For this coursework, a clustering algorithm was initially employed using scikit-learn’s DBSCAN function. A PCA was performed to reduce the dimensionality down to two, enabling data visualisation. **Figure 1** displays the output of this clustering process, revealing ten distinct clusters, each represented by a unique colour, where the black points correspond to anomalous data, aligning with areas of data sparsity. Although the results were promising, the presence of numerous clusters made it challenging to establish a single threshold for distinguishing anomalous data points effectively.

A diagram of different colored dots

Description automatically generated

**Figure 1:** Clustering output

As a result, another approach pursued involved leveraging an autoencoder, a neural network architecture capable of learning essential data characteristics. By training the autoencoder to recognise key data features, it became possible to detect the anomaly status of a data point based on its reconstruction error. This approach facilitated the establishment of a single threshold that effectively balanced precision and recall during the evaluation of the training loss associated with data reconstructed by the autoencoder.

**2. Methodology**

The autoencoder architecture was implemented using scikit-learn’s MLPregressor package, featuring an input layer with 52 neurons, a 2-neuron bottleneck layer, and a subsequent output layer consisting of 52 neurons. Tanh activation functions were employed, chosen for their ability to produce both positive and negative output values.

Following the architectural setup, the training and test data underwent standardisation, a necessary pre-processing step to scale data attributes to a consistent range. Then the autoencoder was trained on the standardised training data, and the reconstructed data was generated. To assess the performance of the autoencoder, the reconstruction error, quantified as the mean absolute error between the input data the autoencoder output was computed.

To evaluate the autoencoder's effectiveness visually, the reconstruction error for both normal and anomalous data points was plotted. **Figure 2** shows that the reconstructed training data points in red, match the input data in blue. In **Figure 3**, most of the reconstructed test data points do not match the input data in blue. In conclusion, there is a larger reconstruction error on the anomalous data.

A red and blue dots

Description automatically generated

**Figure 2:** Reconstructed output for training data

A red and blue dots

Description automatically generated

**Figure 3:** Reconstructed output for test data

Plotting histograms of the reconstruction error for normal and anomalous data facilitates the examination of potential threshold choices. Initially, a threshold of one mean plus two standard deviations was chosen, as shown in **Figure 4**. This yielded a 95% training accuracy and 98% testing accuracy. This high accuracy is indicative of overfitting onto the training set, especially since the normal and anomalous set were standardised with their own respective means and standard deviations. In real life, the model would need to detect anomalies from a mixed set of normal and anomalous data, thus the mean and variance of the anomalous data only cannot be singled out.

A screen shot of a graph

Description automatically generated

**Figure 4:** Training loss for normal and anomalous data

Therefore, the decision was made to standardise the test set with the mean of the training set as this is a better representation of real life. To combat overfitting, the regularisation term built into MLPregressor was set to a higher value. Cross validation on the training set to obtain the threshold would have reduced overfitting but this was difficult to implement while adhering to the required function inputs and outputs and time limit specified in the guidelines of the coursework. As a cautious compromise, a lower threshold value of one mean and one standard deviation was chosen.

**3. Pseudocode**

Fit\_preprocess(*data\_path*)

*data ← load data from data\_path*

*data ← drop the 'label' column*

*mean ←* Mean(*data*)

*std ←* Standard\_deviation(*data*)

*preprocess\_params ←* Dict{ *mean, std*}

***return*** *preprocess\_params*

Load\_and\_preprocess(*data\_path, preprocess\_params*)

*data ← load data from data\_path*

*y ← drop the 'label' column in data*

*data ← standardise with mean and std from preprocess\_params*

***return*** *standardized data, y*

Fit\_model(*X*)

*hidden\_layers ← autoencoder architecture*

*model\_layers ← initialize MLPRegressor*

*regr ← fit X onto model*

*decoder\_out ← predict autoencoder transformation of X using the fitted model*

*tl ← compute reconstruction error*

*threshold ←* Mean(*tl*) *+ 2×* Std(*tl*)

*model ←* Dict{*threshold, regr*}

***return*** *model*

Predict(*X, model*)

*decoder\_out ← predict autoencoder transformation of X using model[‘regr’]*

*tl ← compute reconstruction error of X*

*limit ← model['threshold']*

*y\_pred ← array of zeros of* Len(*tl*)

***for*** *i* ***in***Range(Len(*tl)*)

***if*** *tl[i] > limit* ***then*** *y\_pred[i] ← 1*

***else*** *y\_pred[i] ← 0*

***return*** *y\_pred*