Anomaly Detection
Lab 3 - Isolation Forest, LODA

Andrei Hîji

November 2024

In this lab we will use anomaly detection algorithms like Isolation Forest and LODA.

In order to use **Deep Isolation Forest** model from **pyod** you have to install **pythorch**:

```
pip install torch
```

A few methods that will be useful to you throughout this lab session are:

- **sklearn.datasets.make_blobs** is used to generate isotropic Gaussian clusters (covariance matrix can be represented in this form: $\Sigma = \sigma^2 I$), its parameters include:
  - Number of samples: n_samples=100
  - Number of features: n_features=2
  - Centers and standard deviation of the clusters: centers=None, cluster_std=1.0,
  - Bounding box for each cluster center when centers are generated at random: center_box=(-10.0, 10.0)
  - shuffle=True
  - random_state=None
  - return_centers=False
- **sklearn.model_selection.train_test_split** splits datasets into random train and test subsets and include the parameters:
  - Input data: *arrays
  - Test data size and train data size: test_size=None, train_size=None
  - random_state=None
  - shuffle=True
  - stratify=None - used to split in a stratified fashion
- **sklearn.metrics.roc_auc_score** computes ROC AUC:

  - True labels: y_true
  - Obtained scores: y_scores

- **scipy.io.loadmat** loads data from a MATLAB file and includes the following parameters:
  - file_name
  - Dictionary in which to insert matfile variables: mdict=None
  - appendmat=True : to append the .mat extension to the end of the given filename

- **pyod.utils.utility.standardizer** transforms data to zero-mean and unit variance; it includes the parameters:
  - Training samples and test samples: X, X_t=None

- **numpy.quantile** computes the q-th quantile of the data along the specified axis (the value below which the specified percentage of data falls); its parameters include:
  - Input array: a
  - Probability or sequence of probabilities of the quantiles to compute. Values must be between 0 and 1 inclusive: q

- **numpy.random.uniform** draws samples from a uniform distribution; its parameters include:
  - Lower boundary and Upper boundary of the output interval : low, high
  - Output shape: size

- **numpy.histogram** computes the histogram of a 1D dataset; its parameters include:
  - Input data : a
  - Number of equal-width bins in the given range (if it's an int) or a monotonically increasing array of bin edges (if it's a sequence)
  - The lower and upper range of the bins: range

1 IForest main ideas

Anomaly Score:

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

- $E(h(x))$ - average of $h(x)$ from all the iTrees of the IForests

- $c(n)$ - average path length of an iTree (average path length of unsuccessful search in BST)

Main parameters:

- sub-sampling size $\psi = 256$ (height limit $l = 8$)
- the ensemble size $t = 100$

2 Exercises

2.1 Ex. 1

1. In the first exercise you will design a simpler variant of LODA. First you will generate a 2D dataset that follows a standard normal distribution (500 points) using **sklearn.datasets.make_blobs**.

2. Then you will randomly generate 5 unit-length projection vectors (you can use **numpy.random.multivariate_normal** with (0, 0) mean and identity matrix as covariance matrix) that will be used to generate 1D histograms (for the projected values). You will compute the corresponding histograms with equal-width bins using **numpy.histogram** (for the **range** parameter use a larger interval than the range of the projected values). For each histogram compute the probability corresponding to each bin and use them to compute the anomaly score of a sample as the mean of the probabilities (corresponding to each histogram).

3. For testing, generate a dataset with 500 points from a uniform distribution (between -3 and 3 using **np.random.uniform**). Plot the points in the test dataset using a colormap (related to the anomaly scores).

4. Use different number of bins and see how this affects the score map.

2.2 Ex. 2

1. In this exercise we will try to see how the standard Isolation Forest algorithm introduces some artifacts when computing the anomaly scores. You will generate 2 clusters of 2-dimensional data using **make_blobs()** function. The 2 clusters will have **(10, 0)** and **(0, 10)** as centers, **1** as standard deviation and **500** samples each.

2. You will fit an **IForest** model (from **pyod.models.iforest**) using this data (and a contamination rate of **0.02**). Test data will be generated from a uniform distribution over the interval **(-10, 20)** using **np.random.uniform** and will contain **1000** samples.

3. Find the anomaly scores for the test data and plot the samples using a colormap (related to the anomaly scores). Observe the artefacts introduced by the axis-parallel separating hyperplanes used by standard IForest.

4. Repeat the same procedure for Deep Isolation Forest model (**DIF** from **pyod.models.dif**) and **LODA** (from **pyod.models.loda**) and use 3 subplots for the 3 figures.

5. Try different number of neurons for the hidden layers used by **DIF** and different number of bins for **LODA**. Try to explain why the score maps for **LODA** look that way.

6. Redo all the steps in 3D (use **(0, 10, 0)** and **(10, 0 , 10)** as centers for the two clusters).

2.3 Ex. 3

1. For this exercise we will need the shuttle dataset from ODDS (`https://odds.cs.stonybrook.edu/shuttle-dataset/`). Load the data using **scipy.io.loadmat()** and use **train_test_split()** to split it into train and test subsets (use 40% of data for testing). Normalize your data accordingly.

2. Fit IForest, LODA and DIF using the training data and compute the balanced accuracy (**BA**) and the area under the curve (**ROC AUC** - using **sklearn.metrics.roc_auc_score**) for each model. Compute the mean **BA** and **ROC AUC** obtained for 10 different train-test splits for each of the models.