Anomaly Detection
Lab 2 - Basic anomaly detection algorithms

Andrei Hîji

October 2024

In this lab we will use some basic anomaly detection algorithms like leverage scores, KNN and LOF.

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

- **pyod.utils.data.generate_data_clusters** is used to generate clusters of synthetic data, its parameters include
  - Number of training and test samples: n_train=1000, n_test=500
  - Number of clusters and features: n_clusters=2, n_features=2
  - Contamination level: contamination=0.1
  - Characteristics of the clusters: size='same', density='same', dist=0.25
  - random_state=None
  - return_in_clusters=False
- **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

- **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
- **pyod.models.combination.average** takes the average of the outlier scores; uses the following parameters:
  - Score matrix of shape (n_samples, n_estimators): scores
  - estimator_weights=None - for weighted average
- **pyod.models.combination.maximization** takes the maximum score and includes the parameter:
  - Score matrix of shape (n_samples, n_estimators): scores
- **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

1 Exercises

1.1 Ex. 1

In this exercise we verify numerically the purpose of the leverage scores discussed in class. Generate a random linear model $y = ax_1 + b + \epsilon$ where ϵ is random Gaussian noise with mean μ and variance σ^2. For various values of μ, σ^2 generate data and compute the leverage scores for all the points. Create four types of points: regular (low noise, close to the model), high variance on x, high variance on y and finally high variance on both x and y. Plot all results, group in subplots for the various values of noise variance. Mark on the plots the points with the highest leverage scores. Repeat the exercise for the 2D case $y = ax_1 + bx_2 + c + \epsilon$.

1.2 Ex. 2

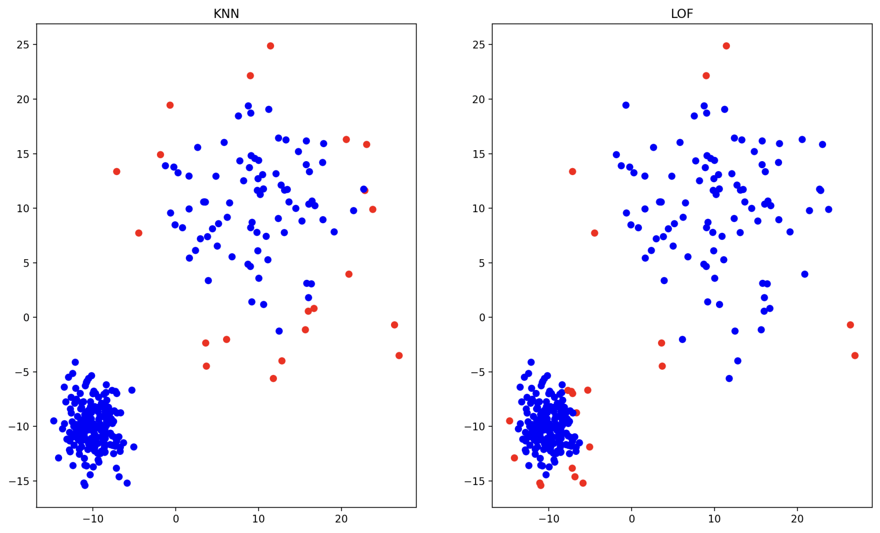
Using the function **generate_data_clusters** generate a 2-dimensional dataset with 400 train samples and 200 test samples that are organized in 2 clusters, with **0.1** contamination. Train a **KNN** model from **pyod.models.knn**. Use 4 subplots in order to display using different colors (for inliers and outliers):

- Ground truth labels for training data
- Predicted labels for training data
- Ground truth labels for test data
- Predicted labels for test data

Use different values for the **n_neighbors** parameter and observe how this affects the detection of small clusters of anomalies. Also compute the balanced accuracy for each parameter.

1.3 Ex. 3

In this exercise we will see the limitations of distance based algorithms like **KNN**. Specifically, we will observe how **KNN** behaves when our data clusters have different densities and how **pyod.models.lof.LOF** solves the problem by considering the variations of the local densities of the datapoints. First, generate 2 clusters (**200** and **100** samples respectively) with 2-dimensional samples using **(-10, -10)** and **(10, 10)** as centers, **2** and **6** as standard deviations using **sklearn.datasets.make_blobs()** function. Then, fit **KNN** and **LOF** with the generated data using a small contamination rate (0.07) and find the predicted labels. Use 2 subplots to plot (using different colors for inliers and outliers) the 2 clusters and observe how the 2 models behave for different **n_neighbors**.



1.4 Ex. 4

For this exercise we will need the cardio dataset from ODDS (`https://odds.cs.stonybrook.edu/cardiotocogrpahy-dataset/`). Load the data using **scipy.io.loadmat()** and use **train_test_split()** to split it into train and test subsets. Normalize your data accordingly. You will use an ensemble of classifiers of the same type (**KNN** or **LOF**) in order to create an average/maximization strategy (average/maximum score will be returned). Create 10 **KNN/LOF** models for which you vary parameter **n_neighbors** from 30 to 120 (here you can use other intervals/steps if you observe that they produce better results). Fit each model, print the balanced accuracy (BA) for train/test data and store both the train and test scores in order to use them later. Normalize both scores using **pyod.utils.utility.standardizer()** and use **pyod.models.combination.average()** and **pyod.models.combination.maximization()** to find the final scores for the 2 strategies. For each of them find the threshold used for classification (using **numpy.quantile()** with the known contamination rate of the dataset), compute the predictions and print the balanced accuracy (BA).