Assignment 4

Anomaly Detection: practical issues (Python version)

The objective of this assignment is to learn the usage of anomaly detection algorithms and tackle practical issues in this process. You will start with a straightforward approach, just applying two anomaly detection algorithms on the simple synthetic data and observing the performances you would obtain in this case. Your objective is also to test how much the performances are affected when you change the parameters of these algorithms.

You will use PyOD library to complete the tasks of this assignment. Please see the documentation for installation here: https://pyod.readthedocs.io/en/latest/install.html

Your final task involves applying your knowledge on a real-world anomaly detection dataset. You will test several algorithms using different evaluation metrics and study which approach is more suitable for your data.

TASK 0: warm-up) Read the data stored in anomaly_data.csv file. This file includes a simple synthetic dataset containing normal and anomalous samples. You can start by using pandas *read_csv()* function. You should be familiar with data importing from previous assignments.

Briefly investigate the dataset by visualizing the normal and anomaly samples in a 2D plot. You can use *"matplotlib"* to make plots. You can visualize samples from different classes with different colors using their label information. Note that label "0" corresponds to normal samples, while label "1" shows anomalies.

Hint: You can use the following code to select subset of columns.

```
X, y = dataframe.iloc[:, :2], dataframe.iloc[:, -1]
```

TASK 1: KNN-based Anomaly Detection

In this task, you will learn how to apply a KNN-based anomaly detection method, which is a proximity-based algorithm that assesses anomalies on the basis of distances to nearest-neighbors.

First import functions that are going to be used in this task using the following code.

```
from pyod.models.knn import KNN
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score,
average_precision_score
from pyod.utils.example import visualize
```

Make sure to look at the documentation on page https://github.com/yzhao062/pyod/blob/master/pyod/models/knn.py This implementation of

KNN outlier detection method can compute outlier scores using different techniques such as largest, mean, and median nearest neighbor distance. It is also possible to specify different distance metrics (e.g., “minkowski”, “euclidian”).

Start splitting data for training and test by using the function below:

```
train_test_split(X, y, test_size=0.3, random_state=0)
```

this function returns training data, test data, training labels, and test labels. Note that outlier detection is an unsupervised learning task. We will not use training labels in our model and only use test labels for evaluation.

For creating the model use

```
clf = KNN(n_neighbors=10, method=”largest”,
metric="euclidean")
```

After training your model with X_train, you can get predictions on X_test using the following

```
y_test_scores = clf.decision_function(X_test)  # outlier
scores
y_test_pred = clf.predict(X_test)  # outlier labels (0 or 1)
```

The first function computes outlier scores for each sample, where the higher the score, the more likely the sample is an anomaly. The second function converts outlier scores into predicted labels (outlier vs normal), using a threshold based on the level of contamination in data (percentage of outliers) in the dataset.

Report performance using two different evaluation metrics “ROC-AUC” and “Average Precision”. Use following functions

```
roc_auc_score(y_test, y_test_scores)
```

```
average_precision_score(y_test, y_test_scores)
```

Generate the visualizations using the visualize function.

```
visualize(“KNN”, X_train, y_train, X_test, y_test,
y_train_pred,
          y_test_pred, show_figure=True, save_figure=False)
```

TASK 2: Parameter sensitivity

In this task, we will inspect the performance of the KNN algorithm w.r.t. K and the outlier scoring method. You can specify different values for n_neighbors = [2, 4, 6, …, 20] and method = {"largest", "mean", "median"}. Write two loops to try every value of K and method. This will give you an exhaustive search strategy for tuning hyperparameters of KNN outlier detection algorithm.

Report the best "n_neighbors" and "method" that give the highest "AUC-ROC" and "Average Precision". Discuss how much "n_neighbors" and "method" affect the performance on different metrics.

TASK 3: Local Outlier Factor (LOF)

Now repeat Task 1 using the LOF algorithm. You can easily modify your code for LOF using PyOD library by importing the function below.

```
from pyod.models.lof import LOF
```

LOF also has "n_neighbors" as a parameter. Repeat Task 2 for LOF by varying n_neighbors = [2, 4, 6, …, 20]. Compare and discuss the parameter sensitivity of KNN and LOF algorithms on different number of nearest neighbors.

TASK 4: Real-world application of Anomaly Detection

This last task is intended to be performed independently, so we only provide limited instructions. The knowledge acquired during the lectures and from the assignments should be sufficient for you to set up this process from the beginning to the end.

You will use the data cardio.csv which consists of measurements of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms classified by expert obstetricians. Anomalies represent the samples diagnosed as pathologic.

The objective of the task is to apply and compare different anomaly detection methods on this real-world healthcare data and compare the results using different evaluation metrics. We suggest you pick "three" **unsupervised** algorithms from PyOD library. You can choose the algorithms that were specifically taught in the lectures or prefer to try and learn new methods. We suggest you include one algorithm that has been mentioned during the lectures such as LOF, Isolation Forest, or One-Class SVM.

You will report your results using "ROC-AUC" and "average precision". We suggest you to not tune the hyperparameters of your methods and instead use their default setting. Anomaly detection is an unsupervised problem, and it is often not easy to select the best parameters in practice.

Try to understand the differences between your algorithms' performances and also the differences between the results of different evaluation metrics.

TASK 5 (OPTIONAL): Improving Detection Performance

In this task, you will try strategies to improve your algorithm's detection performance obtained on the previous task. You can try different data preprocessing and feature selection techniques and see whether it is possible to obtain improved results.

CONCLUSION

With this assignment, you have used a number of methods to detect anomalies in data. The assignment included applying some common techniques to both synthetic and real-world dataset as well as different evaluation strategies.

It is important to remember that there is "no free lunch" in anomaly detection. An algorithm that works very well in a certain dataset, may not work that well in another. Therefore, it is very important to understand the data and the application domain to decide which algorithm suits your problem better.

Here are a few additional questions, so that you can reflect on what you have learned in this assignment. You can think about them yourself, and also discuss them with your course and group mates.

- Which anomaly detection algorithm is more suitable for detecting outliers in the synthetic dataset considering different evaluation metrics and parameter sensitivity?
- What are the key differences between ROC-AUC and AUC-PR?
- Which algorithm is suitable to detect anomalies in the cardiotocograms dataset? Is there a significant difference in the performances considering different evaluation metrics? Are your findings different from those of the previous dataset?