about this liveProject

This liveProject aims to introduce you to the anomaly-detection techniques available in scikit-learn. You will deal with a medical dataset because such data is often skewed. Datasets with skewed classes provide good testbeds for anomaly-detection algorithms. There are many rare conditions, and it is always easier to collect a sample of people who do not suffer from a particular illness than it is to find people with this problem.

In this project, you will work on creating a model to detect whether patients referred to a clinic have abnormal thyroid function. Such a model could alleviate the pressure on a hospital by helping to assign resources to the most important cases. Automated diagnosis is one of the many use cases of machine learning in the medical industry.

Techniques employed

Anomaly and outlier detection is a very special problem in data science. The goal of outlier detection is to identify instances seen as “anomalous.” These are instances not conforming to some standard of what the average case in a dataset looks like. Anomalies/outliers by definition are few and far between. This makes it a particularly difficult problem, as in some cases, traditional supervised algorithms might not work very well.

The goal of this project is to familiarize you with various techniques for anomaly detection using the scikit-learn library. You will use different approaches for discovering anomalies using an example dataset from the medical domain. You will also learn how to measure the performance of anomaly detection algorithms by adapting the more familiar evaluation metrics typically used in supervised binary classification.

- Learn how to load MATLAB data in Python.
- Use scikit-learn to run outlier-detection algorithms, like isolation forest.

Project outline

This liveProject is divided into 5 milestones with the first milestone for setting up the data and the remaining ones for comparing 4 different scikit-learn algorithms.

- Robust covariance
- One-class SVM
- Isolation forest

- Local outlier factors

1. Download and Process the Thyroid Dataset

- Download the Thyroid dataset and check to see if it requires any further cleaning. This dataset is relatively clean, contains a decent number of instances (close to 4000), has small dimensionality, and features a high percentage of anomalies. This makes it ideal as a starting point for algorithm comparison.
    - Download the data from Thyroid Disease dataset.
    - Use `sio.loadmat()` to read the MATLAB file into pandas.

2. Run a One-Class Support Vector Machine (SVM)

- In this milestone, we will kick off setting up the data for prediction tasks. We will copy some functions over from the starter template and edit the template so that it can continue generating probability scores for the test data. This is important for clinical applications because clinicians will be interested in seeing the confidence in addition to the predicted labels.
- You will also learn the nuances between novelty and outlier detection and how to increase performance by converting the problem to a novelty-detection task. Subsequently, your goal will be to run a one-class SVM with a radial basis function (RBF) kernel and a polynomial kernel of degrees 2 and 3. You will record its performance using various classification metrics, such as Average Precision, F1 score, Precision, and Recall.

3. Run Robust Covariance

- In this milestone, we are going to see if running the `EllipticEnvelope()` method as an outlier-detection algorithm gives us better performance when we vary the outliers fraction (1,2 and 4x). This is an interesting algorithm because, although it is supposed to be for data distributions that are Gaussian, in practice it performs reasonably well even if this assumption is not proven *a priori*.

4. Run Isolation Forest

- Isolation Forest is an outlier-detection technique that uses decision trees, a familiar technique. So the main parameter we are going to vary is the total number of trees.

- Try running isolation forests with 50, 100, and 200 trees. How does the performance change?

5. Run the Local Outlier Factor (LOF) Algorithm

- The main parameter in the LOF algorithm is the total number of neighbors. Try running the algorithm with 3, 10, 20, and 50 neighbors. How does this affect the result? We are going to set the novelty parameter to True so that the algorithm is set up to detect novelties rather than outliers.