CSC 730 Assignment 07
Isolation Forests

Mangesh Sakordekar

Overview

The aim of this assignment is to:

- Write your own version of isolation forest code.
- Run your code on the given dataset to obtain anomalousness scores for each point
- Sort the points by anomalousness scores and generate a precision-recall curve.
- Generate plot showing anomalousness regions from your forest.

Importing Dataset

Data was imported from the provided numpy zip file, The data was then scaled and normalized and compressed using two methods - PCA and tSNE.

Isolation Forest Implementation

Node Class

Node class holds the information of each node in the tree. This class can be initialised as an external node or an internal node.

```
class Node:

    def __init__(self, size = 0, left = None, right = None,
                 attr = None, val = None, is_external = False):
        self.size = size
        self.left = left
        self.right = right
        self.split_attr = attr
        self.split_val = val
        self.is_external = is_external


    @classmethod
    def internal(node, left, right, attr, val):
        return node(None, left, right, attr, val, False)

    @classmethod
    def external(node, size):
        return node(size, None, None, None, None, True)
```

Figure 1: Node Class Implementation

Isolation Forest Class Implementation

Isolation Forest class was implemented following the pseudo-code provided in the slides. The important functions in the class are isolation_tree and path_length.

The isolation_tree function generates individual isolation trees using the set class parameters. To form the tree, the function randomly selects a feature and a filter value within the min and max values of the feature column. Then it splits the dataset into two, one with values lesser than the filter value and one with values grater than the filter value. It calls itself recursively to form a binary tree structure. The stopping criteria is if the height limit is reached or the size of the dataset is less than or equal to 1. In this implementation, the default value for the sampling size is 256 and number of trees is 100. All the runs used in this project were done with the default parameters.

```
def _isolation_tree(self, data,counter=0):
    if (counter == self._height_limit) or len(data)<=1:
        return Node.external(len(data))

    feature_index = random.randint(0, data.shape[1]-1)
    min_val = np.min(data[:,feature_index])
    max_val = np.max(data[:,feature_index])

    filter_value = random.uniform(min_val, max_val)

    left_indexes = np.where(data[:,feature_index] < filter_value)[0]
    right_indexes = np.where(data[:,feature_index] >= filter_value)[0]

    return Node.internal(self._isolation_tree(data[left_indexes,:], counter + 1),
                         self._isolation_tree(data[right_indexes,:], counter + 1),
                         feature_index, filter_value)
```

Figure 2: Isolation Tree Implementation

The path_length function calculates the path length for each given data point. Depending on the filter value of the node it directs the control to the left or the right sub-tree until it reaches an external.

```
def _path_length(self, x, tree, curr_length):
    if tree.is_external :
        return curr_length + self._c_factor(tree.size)
    if x[tree.split_attr]  < tree.split_val:
        return self._path_length(x, tree.left, curr_length+1)
    return self._path_length(x, tree.right, curr_length+1)


def _get_paths(self, data_point):
    paths = []
    for tree in self._forest:
        paths.append(self._path_length(data_point, tree, 0))
    return paths
```

Figure 3: Path Length Implementation

Running Isolation Forest

Using PCA'd data

An instance of the Isolation Forest class was fit on PCA'd version of the data. The anomaly scores of the same dataset were obtained and stored. Then a contour plot was plotted to visualize the decision boundaries of the model.

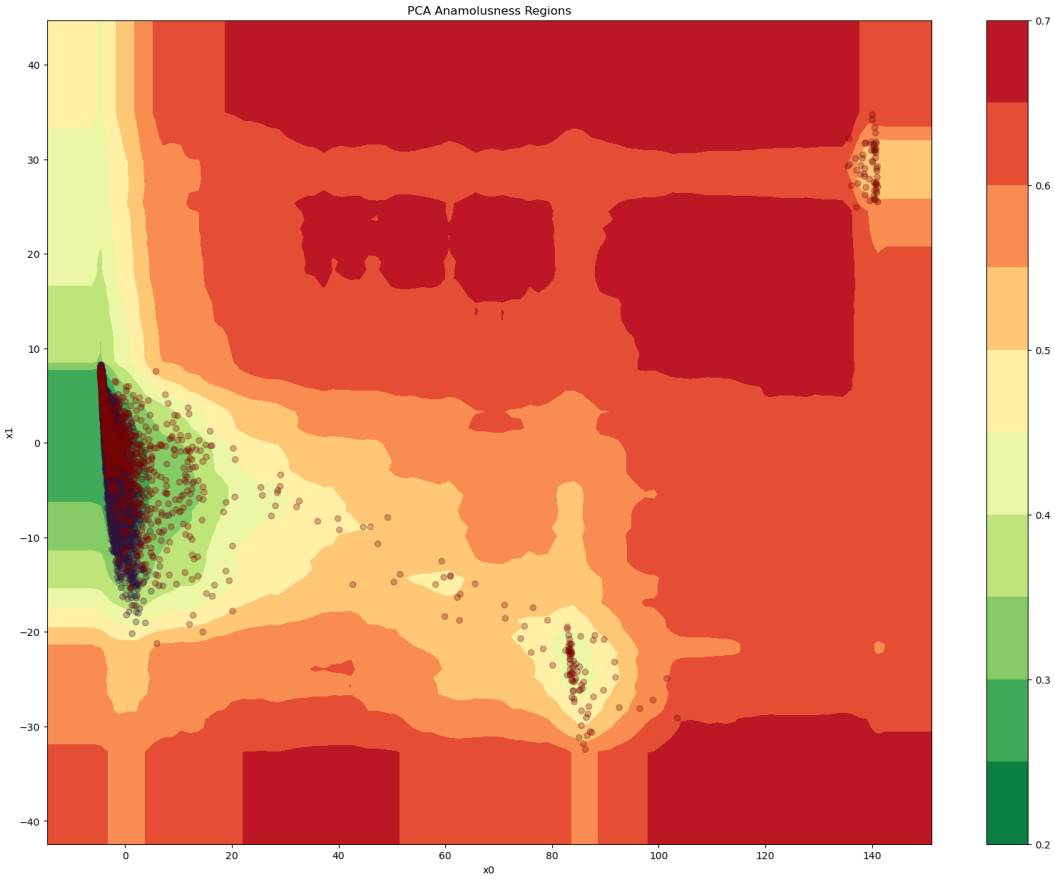


Figure 4: Decision Boundaries of Isolation Forest fitted on PCA'd Data

In this plot, the green regions mark the normal space and the red regions mark the anomalous regions. We can also see that the blue dots, which are the normal data points lie in the normal region and the red dots lie in the anomalous regions. Looking at the boundaries, this isolation forest classifies the data pretty well.

Using tSNE'd data

An instance of the Isolation Forest class was fit on tSNE'd version of the data. The anomaly scores of the same dataset were obtained and stored. Then a contour plot was plotted to visualize the decision boundaries of the model.

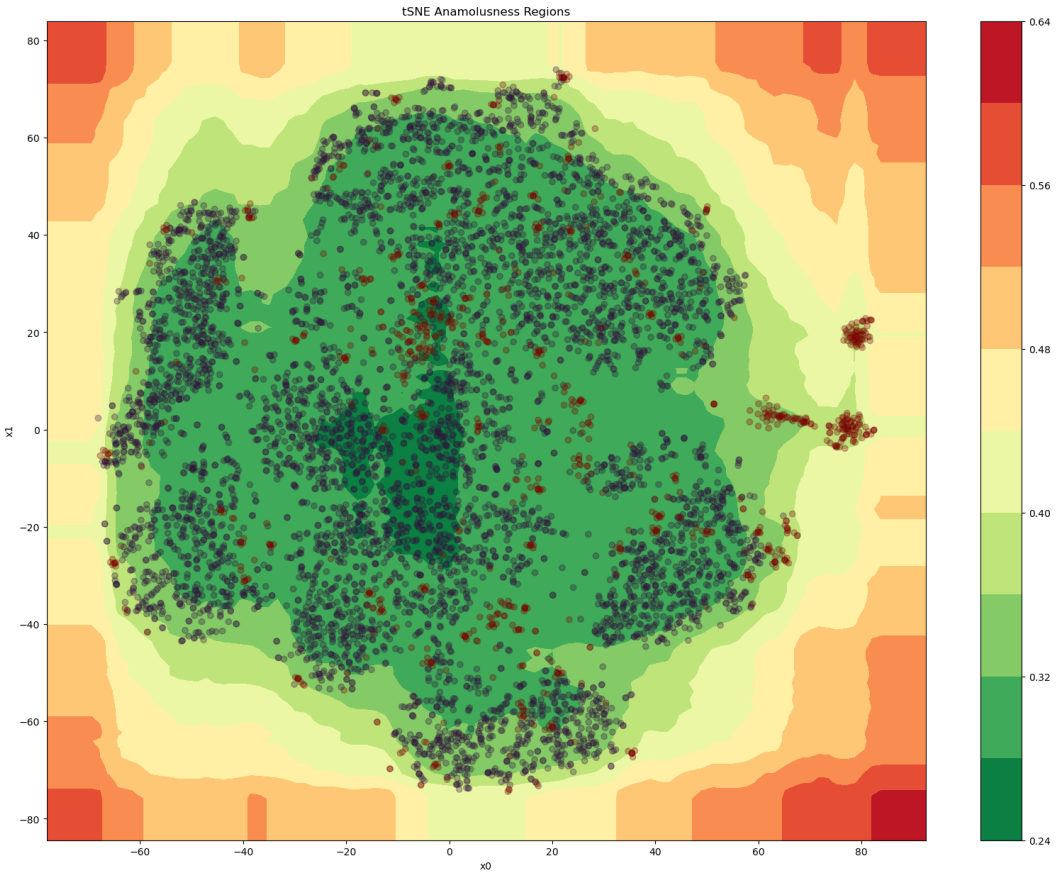


Figure 5: Decision Boundaries of Isolation Forest fitted on PCA'd Data

In the tSNE'd data there is a major overlap between the normal and anomalous points, thus making it difficult for the algorithm to find the anomalies, making this a poor classifier.

Uncompressed Dataset

The Isolation forest was also fitted using uncompressed dataset. The scores were generated and stored to plot ROC and PR curves.

Metrics

To compare the performance of the implementation of the Isolation forest, sklearn's implementaion was imported and fitted on the three datasets. The scores from these runs were used to plot ROC and PR curves.

Receiver Operating Characteristic (ROC) Curve

ROC curve plots the True Positive Rate against the False Positive Rate. The plot below shows 6 ROC curves. Red, blue and green lines are plots for PCA'd data, tSNE'd data and uncompressed data respectively using my implementation of Isolation forest. Orange, purple and yellow lines are plots for PCA'd data, tSNE'd data and uncompressed data respectively using sklearn's implementation of Isolation forest.

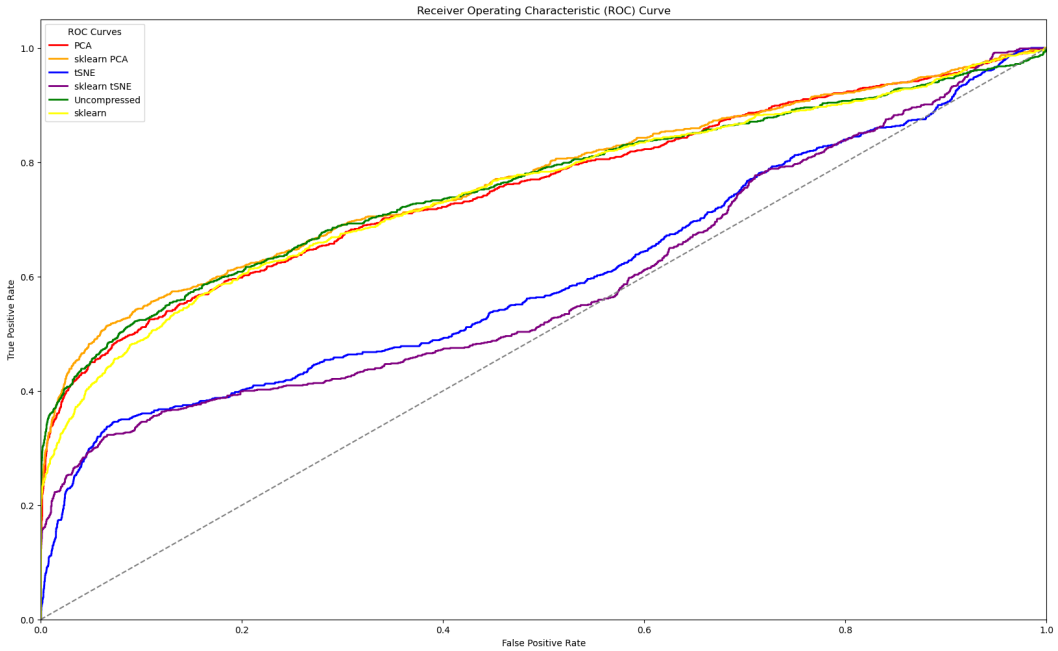
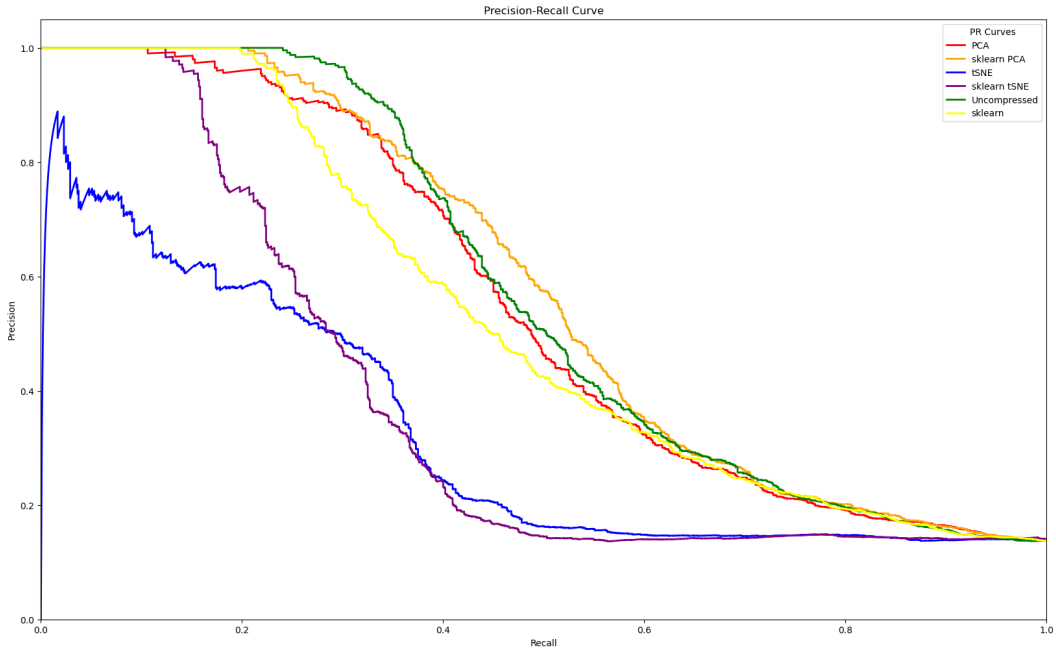


Figure 6: ROC Curve

Precision-Recall Curve

Precision Recall curve plots the Precision against the True Positive Rate also known as Recall. Red, blue and green lines are plots for PCA'd data, tSNE'd data and uncompressed data respectively using my implementation of Isolation forest. Orange, purple and yellow lines

are plots for PCA'd data, tSNE'd data and uncompressed data respectively using sklearn's implementation of Isolation forest.



Results

Looking at the plots, it can be seen that the my implementation of Isolation Forest works as good as sklearn's implementation of the same algorithm. Though the sklearns implementation performs better on the tSNE'd data. Among the datasets, the algorithms prefer the uncompressed and the PCA'd versions of the data but struggle with the tSNE'd data. We can also see the decision regions for the PCA's and tSNE'd data in the contour plots which shows the overlap between the anomalies and normal data-points leading to the poor performance of the model.

Conclusion

Isolation Forest was successfully implemented from scratch and it can handle varying number of features. The plots also showed that this algorithm works as well as the sklearn's implementation.