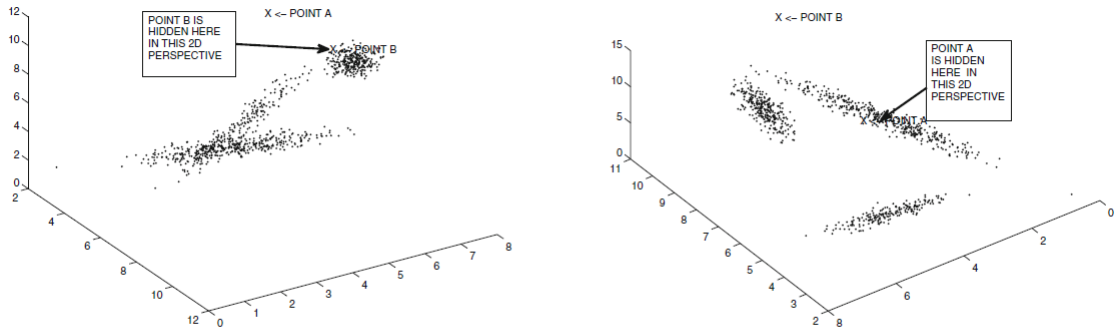
Anomaly Detection with Graph Embedding Ensemble

1 Problem definition

Anomaly Detection (Outlier Analysis) is a common problem in machine learning area like Credit Card Fraud Detection, Factory machine problem and also in image, nlp problem. And there is no solution which can solve or find all outlier. We have to consider the various approach to solve each problem.

In unsupervised anomaly detection, it is more difficult than supervised problem. There are several reason which makes it difficult to find anomaly points in unsupervised problem. One of the reasons is showed in Figure1. Anomaly scores are different which veiw is used to score anomaly points. In this case we usally do ensemble approach to consider various view in finding anomaly points. In this project I also do ensemble approach.



(a) Point 'A' is exposed and 'B' is hidden in this 2d view

(b) Point 'B' is exposed and 'A' is hidden in this 2d view

Figure 1: Various anomaly point in each view

One of the main algorithm in unsupervised anomaly detection is considering the similarity between data points. Normal data point and anomaly data point will have low similarity. Normal data poins have high similarity each other. And also anomaly data points have high similarity each other. So if we can catch this relationship very well we can find anomaly data points.

So two points that I want to solve is *considering various views* and *catching the relationship well.* To catch the similarity relationship I suggest the graph embedding preprocessing. After making graph using raw data, I do graph embedding using that graph. Then embedding vector has relational information about similarity between data points. And do anomaly detection algorithm using this embedding vector. Ensemble approach by making various graph can consider various views.

And graph embedding vector also can be used in superviesed problem. We usually do feature engineering to make our model predict well in machine learning. Graph embedding vector has some feature engineered information from raw data. So using this vector as a new input, we can expect that our model predict more well.

2 Algorithm

2.1 Anomaly Detection in Unsupervised case

I consider only tabular data. But big size data does not fit well in this method. Because we make graph using data-wise similarity, it takes $O(n^2)$ time and memory in calculating similarity. Also high dimension data also does not work well. Because similarity in high dimension does not formed well as we expected (curse of dimension). Let's see the algorithm.

Algorithm

1. Make Adjacency matrix using the each data points similarity.
2. Predefined threshold makes Adjacency matrix to have 0 or 1.
    - if similarity $>$ threshold : 1
    - else : 0
3. Make graph using Adjacency matrix.
4. Do graph embedding algorithm and get embedding vectors.
5. Calculate Anomaly score using embedding vectors with anomaly detection algorithm.
6. With high anomaly score, we define it as an anomaly data point.

Anomaly data points usually have lower similarity with other normal data points. So anomaly node has little edges with normal node but may have edegs with other anomaly node. We can catch the abnormaly different property between data points more well than the raw data.

One more good thing is we can do ensemble by making various graph. When we make graph, various predefined threshold can be used. By doing that we can view the data points in various aspects. In Figure 2 (Yellow is anomaly daeta point), we can see the different embedding result. With different graph, I calculate the scaled anomaly scores in each graph. These scores can be averaged and finally detect the anomaly data points. The points that has large anomaly score are predicted as anomaly data points. In Figure 3, Yellow points are the anomaly points and red line is threshold. As we can see it is quite difficult to find anomaly data points in unsupervised case.

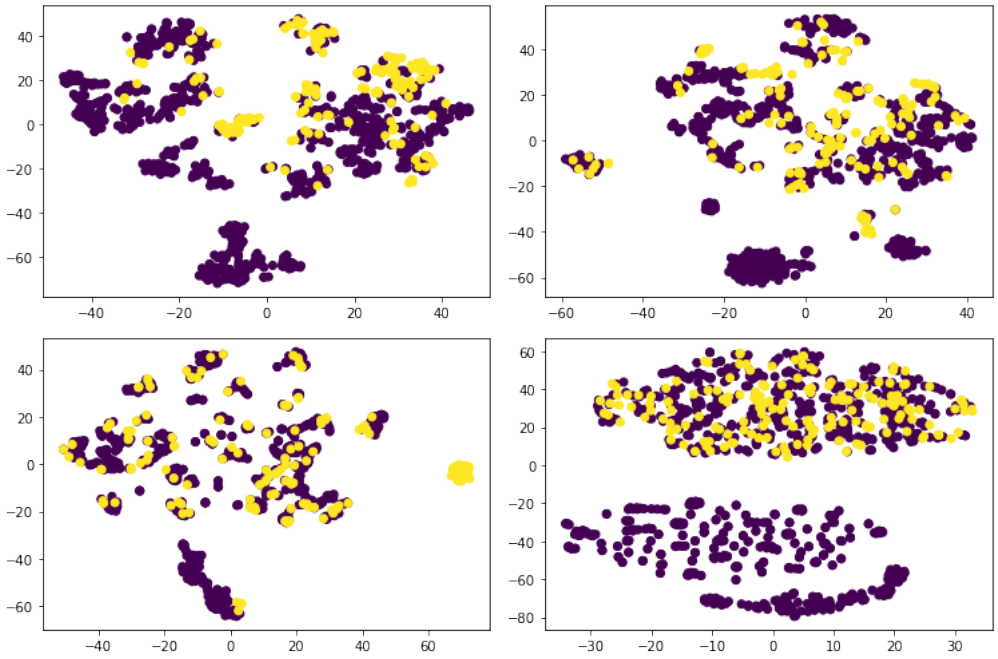


Figure 2: embedding vectors with various cosine similarity threshold 0.3, 0.5, 0.7, 0.9 (from left up to right down)

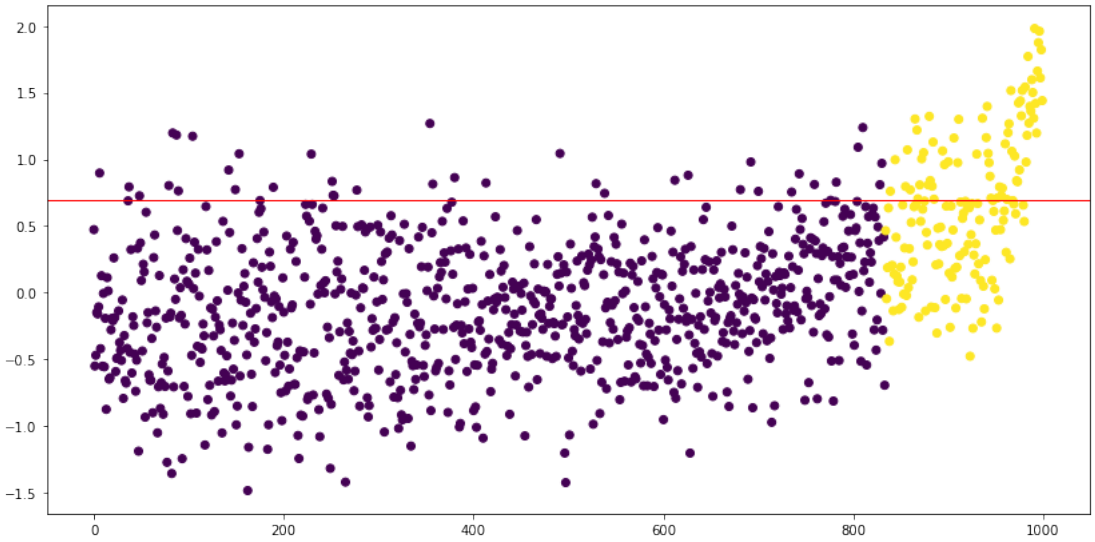


Figure 3: Anomaly score result example (PCA)

2.2 Anomaly Detection in Supervised case

It is simple approach in supervised case. I make embedding vectors with the same approach above. And final anomaly scores are used as input data in ML prediction model. Experimental results will show whether it is effective in prediction modeling.

3 Experimental results

3.1 Anomaly Detection in Unsupervised case

I compared with raw data and graph embedding ensemble approach. All experiment calculated 10 times to avoid randomness and metric is averaged.

As we can in Table 1 and 2, graph embedding ensemble approach is more efficient than using only raw data. But when the imbalance ratio is high graph embedding ensemble approach does not work well. Because num of anomaly data is too small and there is high chance to have similar property with normal data.

- Used algorithm and method
  - make graph with cosine similarity (threshold : [0.3, 0.5, 0.7, 0.9] → Ensemble)
  - graph embedding algorithm : Node2Vec (embedding dimension : 8)
  - anomaly detection algorithm : Mahalanobis, PCA, LOF
- Used data
  - Kaggle Credit Card Fraud Detection (Imbalanced ratio 1:5, 1:10, 1:100)

	Accuracy	ROC AUC	Recall	Precision	F1 score
Raw Malhalanobis	0.835	0.525	0.06	0.526	0.108
Raw PCA	0.838	0.522	0.048	**0.667**	0.09
Raw LOF	0.792	0.516	0.102	0.224	0.14
Graph Malhalanobis (d=8)	0.877	0.512	0.067	0.133	0.089
Graph PCA (d=8)	0.865	**0.724**	**0.512**	0.612	**0.557**
Graph LOF(d=8)	**0.899**	0.504	0.022	0.133	0.038

Table 1: Credit Card Fraud Imbalance 1:5

	Accuracy	ROC AUC	Recall	Precision	F1 score
Raw Malhalanobis	**0.918**	0.579	0.167	**0.682**	0.268
Raw PCA	0.911	0.531	0.067	0.545	0.119
Raw LOF	0.882	0.575	0.2	0.281	0.234
Graph Malhalanobis (d=8)	0.905	0.582	0.189	0.436	0.264
Graph PCA (d=8)	0.915	**0.803**	**0.667**	0.522	**0.585**
Graph LOF(d=8)	0.787	0.462	0.067	0.044	0.053

Table 2: Credit Card Fraud Imbalance 1:10

	Accuracy	ROC AUC	Recall	Precision	F1 score
Raw Malhalanobis	0.974	**0.932**	**0.889**	**0.242**	**0.381**
Raw PCA	**0.983**	0.551	0.111	0.1	0.105
Raw LOF	0.928	0.743	0.556	0.068	0.122
Graph Malhalanobis (d=8)	0.979	0.659	0.333	0.158	0.214
Graph PCA (d=8)	**0.983**	0.606	0.222	0.167	0.19
Graph LOF(d=8)	0.776	0.502	0.222	0.009	0.18

Table 3: Credit Card Fraud Imbalance 1:100

3.2 Anomaly Detection in Supervised case

- Data shape = (1000, 4), Imbalnce ratio = 1:5

- Classifier : Random Forest
- Graph embedding algorithm : Node2Vec
  - cosine similarity threshold : [0.7, 0.9]
  - embedding dimension : 3

	Accuracy	ROC AUC	Recall	Precision	F1 score
raw	0.897	0.832	0.744	0.58	0.652
use embedding	**0.93**	**0.914**	**0.892**	**0.66**	**0.759**

Table 4: Using embedding in Supervised Learning

The data has only 4 columns. It assumes the case that we lack the information(columns). In this case, we usaully do feature engineering. As an one of the feature engineering method, graph embedding method is used in here. When I use graph embedding vectors as inputs, it has better predictability.

4 Discussions

As I expected, graph embedding ensemble method works better than only using raw data. This is because it can catch relationship between data by graph embedding and also consider various views of data structure by ensemble approach. But this method also has cons. It has many hyperparameters and not efficient when raw data is high dimension or high imbalanced case. And also if datea size is large, graph embedding ensemble method will be quite slow and memory consuming.

5 Appendix

- Code : calculate anomaly score example

```
from utils import make_graph, scoring
from node2vec import Node2Vec
from anomaly_prob import anomaly_detection_Malhalanobis
from anomaly_linear import anomaly_detection_PCA
from sklearn.neighbors import LocalOutlierFactor

final_anomaly_score_dim8_mal = np.zeros((len(X),))
final_anomaly_score_dim8_pca = np.zeros((len(X),))
final_anomaly_score_dim8_lof = np.zeros((len(X),))
clf = LocalOutlierFactor(n_neighbors=5)
cosine_threshold = [0.3, 0.5, 0.7, 0.9]

for threshold in cosine_threshold:
    graph = make_graph(X, threshold)
    node2vec = Node2Vec(graph=graph,
                        dimensions=8,
                        walk_length=8,
                        p = 0.5,
                        q = 1.5,
                        weight_key=None,
                        num_walks=30,
                        workers=2,
                    )
    node2vec_fit = node2vec.fit(window=3)
    embedding_vec = np.array(node2vec_fit.wv.vectors)

    # Malhalanobis
    outlier_score_mal, _ = anomaly_detection_Malhalanobis(embedding_vec)
    outlier_score_mal = (outlier_score_mal - np.mean(outlier_score_mal))\
                / np.std(outlier_score_mal)
    # PCA
    outlier_score_pca = anomaly_detection_PCA(embedding_vec)
    outlier_score_pca = (outlier_score_pca - np.mean(outlier_score_pca))\
                / np.std(outlier_score_pca)
    # LOF
    pred_raw = clf.fit_predict(embedding_vec)
    pred_raw = np.where(pred_raw==1, 0, 1)

    final_anomaly_score_dim8_mal += outlier_score_mal / len(cosine_threshold)
    final_anomaly_score_dim8_pca += outlier_score_pca / len(cosine_threshold)
    final_anomaly_score_dim8_lof += pred_raw
```

- Code : anomaly detection algorithm

```
# Mal
import numpy as np

def anomaly_detection_Malhalanobis(X: np.array):
    X_mean = np.mean(X, axis=0)
    S_inv = np.linalg.inv(np.cov(X, rowvar=False))
    anomaly_score = np.zeros((len(X),))
    for i in range(len(X)):
        anomaly_score[i] = (X[i,:] - X_mean) @ S_inv @\
                           np.transpose(X[i,:] - X_mean)
    anomaly_score = np.sqrt(anomaly_score)
    return anomaly_score

# PCA
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import numpy as np

def anomaly_detection_PCA(X: np.array):
    '''
    X : not scaled data
    '''
    scaler = StandardScaler()
    X = scaler.fit_transform(X)
    pca = PCA()
    pca_result = pca.fit_transform(X)
    anomaly_score = np.sum(np.abs(pca_result) \
                / pca.explained_variance_, axis=1)
    return anomaly_score

# LOF : sklearn
```