ECE 8890: Neural Network – fall 2019 Final Project

Ensemble Generative Adversarial Networks for Anomaly Detection

Wenlong Wu, supervised by Derek Anderson and James Keller

University of Missouri

Abstract

Generative Adversarial Networks (GANs) has achieved great success in modeling data distributions on images. The GANs algorithm is extended on the structure data in this project to model the structure data distributions and then to detect the anomalies that are out of the data distributions. Single GANs architecture has limited ability to approximate the data distributions due to the lack of modes the generator creates. The averaging ensemble method is employed in this project to combine the local spatial data distributions of each single GANs creates. The ensemble GANs, along with the Gaussian Mixture Model (GMM), is run on one synthetic dataset and three benchmark clustering datasets and has shown superior performance on approximating data distributions of structure data and on detecting anomalies than the single GANs and the GMM.

Contents

I. Introduction

Anomaly detection, which is also known as novelty detection and outlier detection, has been well studied over the decades. Many algorithm models have been proposed to solve this problem. There are three main aspects to approach this problem: supervised learning, semi-supervised learning and unsupervised learning. The supervised learning method has achieved great success on classification problem but needs a great amount of labeled data to feed into the algorithm, which is usually unavailable in a certain application such as labeled anomalies in this case. Semi-supervised learning and unsupervised learning are the main approaches to anomaly detection. For semi-supervised learning, a small fraction of data is considered as labeled and the majority of the unlabeled data has only normal samples. Thus, we can use unsupervised machine learning techniques that learn the structure in data to learn some notion of normality.

Generative adversarial networks (GANs) [1] is a powerful framework to learn the generative model and has achieved great success on image generation, image super-resolution and so on [2]. However, the GANs algorithm is mostly applied to image datasets to learn the data distributions of images nowadays. Few research has been done on structure data or tabular data to model the high-dimensional data distributions. So in this project, the GANs algorithm is used to model data distributions on structure data or tabular data and then to detect anomalies that are out of the given data distributions.

The vanilla GANs algorithm has two networks competing with each other: the generator and the discriminator. The well-trained discriminator is served as the base model to detect anomalies. However, the performance of the discriminator is associated with the performance of the generator and the training of the generator is usually unstable to produce samples similar to the real samples. So the averaging

ensemble method is used to stabilize the performance of the generator and the discriminator. The proposed ensemble GANs algorithm has a robust performance on approximating data distributions and is also compared with the Gaussian mixture model in this project.

Gaussian mixture model (GMM) [3] is a popular method widely used to model the data distributions in an unsupervised fashion. The ensemble GANs and the GMM are compared together on one synthetic dataset and three benchmark datasets.

The structure of this project report is organized as follows. Section II briefly discusses GMM, GANs and ensemble GANs. Section III introduces the four datasets used in this project. Section IV represents the experimental study, and Section V summarizes conclusions, lesson learned and future work.

II. Technical Description

II.A. Gaussian Mixture Model

Given a *d*-dimensional vector *x*, a Gaussian mixture probability density function can be written as:

$$p(x) = \sum_{i=1}^{c} w_i p_i(x) \tag{1}$$

where c represents the number of mixture components, the mixture weights w_i satisfy $\sum_{i=1}^{c} w_i = 1$ and $w_i > 0$. Each component density $p_i(x), i = 1,2, \dots, c$ is the probability density function of Gaussian distribution parameterized by a $d \times 1$ mean vector u_i and a $d \times d$ covariance matrix Σ_i. The probability density function is shown as follows,

$$p_i(x) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_i|^{\frac{1}{2}}} exp\{\frac{1}{2}(x - u_i)^T (\Sigma_i)^{-1}(x - u_i)\} \tag{2}$$

It has shown that any continuous probability density function can be approximated with high accuracy by using a sufficient number of Gaussian mixture components and by adjusting their means and covariance as well as the weights [4]. For video background subtraction, it is suggested in [5] to use a fixed number of Gaussian mixture components, usually between 3 and 5, to model the data. However, the number of Gaussian components in different situations should change to describe different data patterns. Too many or too few Gaussian components may decrease the generalization or the accuracy of the algorithm. Thus, determining a suitable number of Gaussian components is of vital importance to the mixture modeling. Some clustering algorithms, such as possibilistic clustering algorithms [6] are often used to help determine the number of Gaussian components in the data.

II.B. Generative Adversarial Networks

The GANs algorithm has two networks competing with each other: the generator G and the discriminator D. The discriminator D tries to distinguish the real samples and the samples generated by the generator G. The generator G tries to “fool” the discriminator D, and learns to generate samples similar to the rea data at the same time. The GANs architecture is shown in Fig 1.

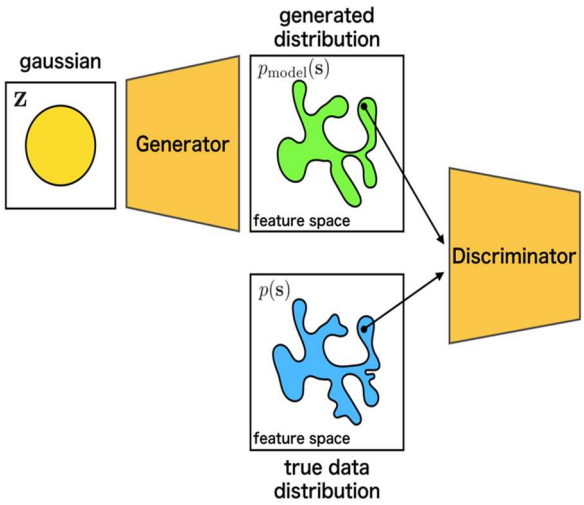


Fig 1. GANs architecture

In this project, the generator and the discriminator are both three-layer multi-layer perceptron (MLP). The hidden layers all have 128 neurons. The input of the generator is a 100-dimensional random noise. The discriminator D and the generator G play a two-player minimax game with value function V (G, D):

$$\min_G \max_D V(G,D) = E_{x\sim p_{data}}[\log D(x)] + E_{x\sim p_z(z)}[log(1 - D(G(z)))\,] \qquad (3)$$

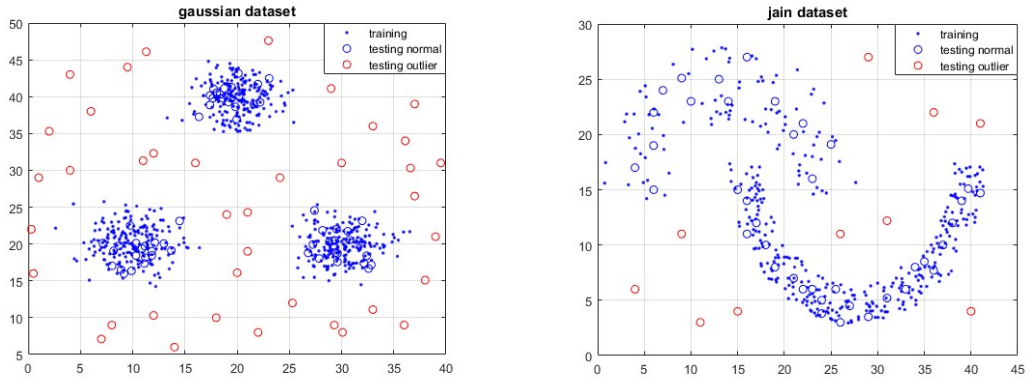
The goal of this function is to maximize the likelihood of the real samples and minimize the likelihood of the samples generated by the generator G.

II.C. Ensemble Generative Adversarial Networks

The discriminator D of the vanilla GANs algorithm learns to distinguish the real samples and the samples generated by the generator G. However, the discriminator may fail to distinguish the data distribution areas where the generator G doesn't go to. So each single GAN is treated as a weak classifier and ensembles together as the final learned data distributions. The ensemble method employed in this project is the averaging operator.

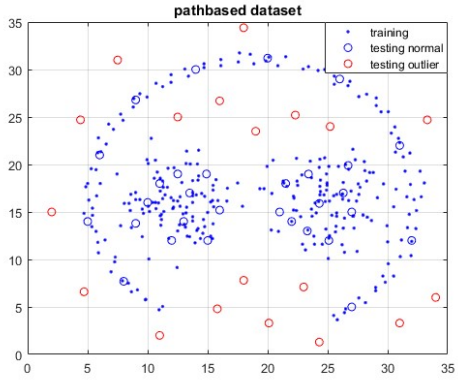
III. Dataset Description

The datasets used in this project consist of one synthetic dataset and three clustering benchmark datasets [7]. I also created some testing sets to evaluate the performance of each algorithm. The visualization plot of the four datasets is shown in Fig 2.

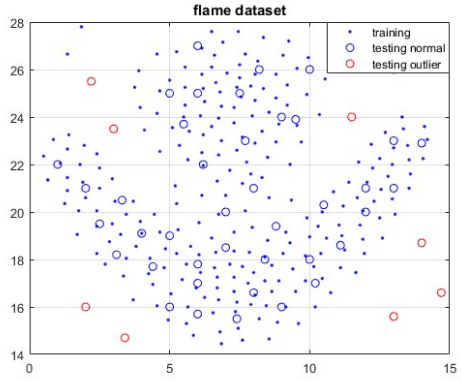


(a) Gaussian

(b) Jain



(c) Pathbased



(d) Flame

Fig 2. Four dataset visualization plot

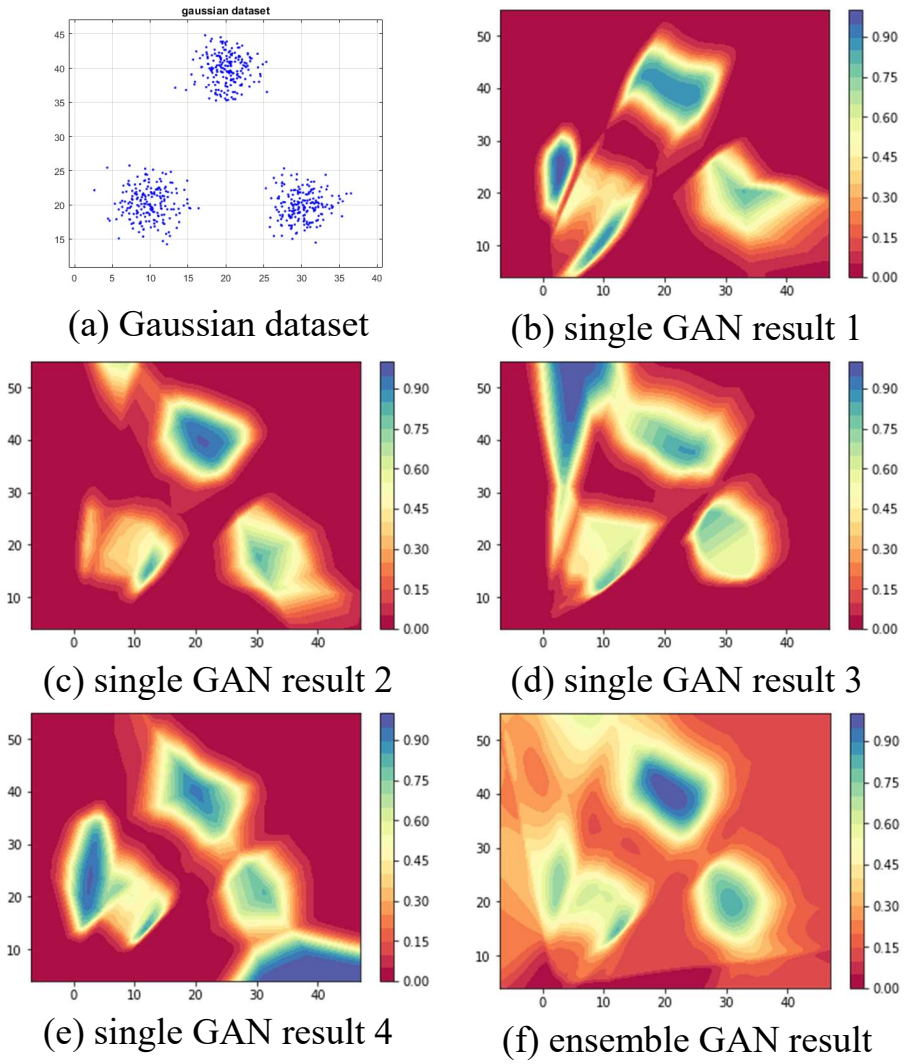
Note that the blue dots in Fig 2. are training samples, the blue circles are normal testing samples and the red circles are anomaly samples.

IV. Experiments and Results

In this section, three experiments were done to illustrate the ideas. The first experiment compares the performance of single GANs and ensemble GANs. The second experiment shows the results of the ensemble GANs on the synthetic dataset and the clustering benchmark datasets. The last experiment extends the ensemble GANs model to a streaming mode and trajectory analysis is conducted to detect trends in the data stream.

IV.A. Experiment 1: Single GANs and Ensemble GANs

In this experiment, single GANs with different random seed was run on the Gaussian dataset, as shown in Fig 3. (a). Four single GAN results with different random seed are shown in Fig 3. (b) – (e). The ensemble GAN which is an averaging ensemble of 10 single GANs is shown in Fig 3. (f).



(a) Gaussian dataset (b) single GAN result 1

(c) single GAN result 2 (d) single GAN result 3

(e) single GAN result 4 (f) ensemble GAN result

Fig 3. Single GAN and ensemble GAN result

As we can see above, the ensemble GANs has better data approximation than each of any single GANs. The reason why the single GAN cannot approximate the real distributions is that the discriminator D cannot distinguish the data distribution areas

where the generator G doesn't go to. The ensemble mechanism used in this case helps to combine the local spatial parts of the real data distributions.

IV.B. Experiment 2: Comparison between Ensemble GANs and GMM on benchmark datasets

The ensemble GANs and GMM are run on the four datasets to compare their performance. The data distributions visualization and testing set results of both algorithms are shown in Fig 4.

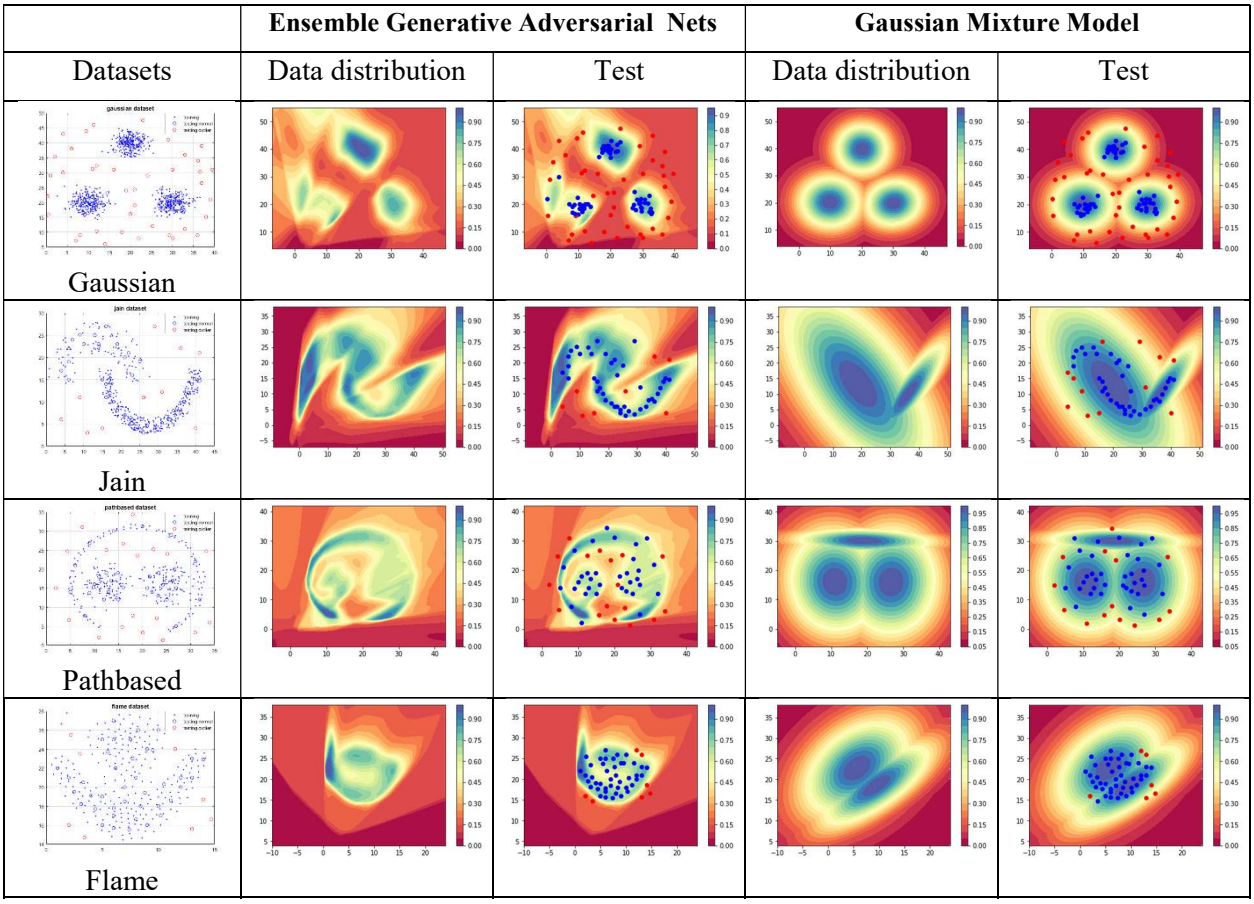


Fig 4. Data distributions visualization and testing set results of ensemble GANs and GMM

The numerical evaluation (precision, recall and F1 score) of the two algorithms are also computed, shown in Fig 5.

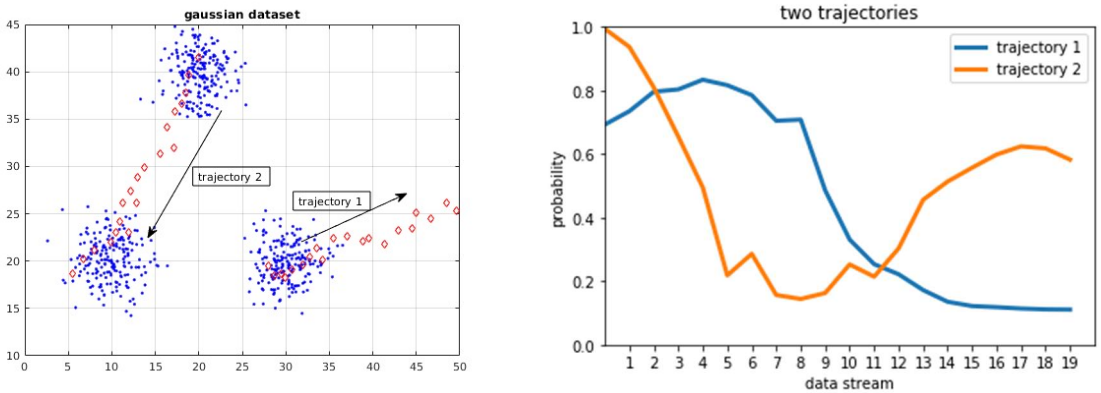
	Ensemble GANs			GMM		
DATASET	Precision	Recall	F1	Precision	Recall	F1
Gaussian	0.92	0.95	0.94	1.00	1.00	**1.00**
Jain	0.84	0.80	**0.82**	0.70	0.90	0.78
Pathbased	0.83	0.80	**0.82**	0.82	0.70	0.76
Flame	0.76	0.70	**0.73**	0.68	0.60	0.64

Fig 5. Numerical evaluation of ensemble GANs and GMM

From both visualization plot figure and numerical evaluation figure, I find the GMM performs perfectly on the Gaussian dataset. It's not surprising at all since the Gaussian dataset is generated from the Gaussian function. For the other three datasets, the ensemble GANs performs better at approximating data distributions than the GMM, especially on some arbitrary shape of clusters.

IV.C. Experiment 3: Trajectory analysis on ensemble GANs

In this experiment, the ensemble GANs is extended to a streaming mode. Two trajectories of the data stream are added to the Gaussian dataset, as shown in Fig 6. (a). Each sample in the trajectory data stream is fed into the trained discriminator D and the output of the data stream trajectory curves is shown in Fig 6. (b).



(a) (b)

Fig 6. Trajectory analysis on the Gaussian dataset

For the first trajectory, the data stream goes from a cluster center to the outside sparse area. The output value of the discriminator decreases as the data stream trajectory goes outside of the cluster. For the second trajectory, the data stream goes from a cluster center to another cluster center. The output value of the discriminator decreases at the beginning and increases as the data stream trajectory reaches the other cluster. Therefore, the streaming version of ensemble GANs has the ability to detect the trends of the data stream.

V. Conclusions, Lessons Learned and Future Work

Generative Adversarial Networks (GANs) has excellent performance on modeling data distributions, especially on images. The GANs is employed to model structure data or tabular data distributions in this project. The generator G and the discriminator D in the GANs compete with each other and grows together. Single GANs has limited ability to model the complex data distributions. The averaging ensemble method is used to combine the local spatial data distributions of single GANs finds. The ensemble GANs performs better at approximating data distributions than the single GANs and shows superior performance on detecting the arbitrary shape of clusters than the GMM. In the future work, some fuzzy integral fusion algorithms can be used to ensemble different single GANs. Besides, the streaming version of ensemble GANs can be revised to update the networks as the data stream comes into the network.

References

[1] Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In Advances in neural information processing systems, pp. 2672-2680. 2014.
[2] Brock, Andrew, Jeff Donahue, and Karen Simonyan. "Large scale gan training for high fidelity natural image synthesis." arXiv preprint arXiv:1809.11096 (2018).
[3] D. Reynolds, “Gaussian Mixture Models”. *Encyclopedia of Biometrics*, pp.827-832, 2015.

[4] C. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[5] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking", *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*.
[6] R. Krishnapuram and J. M. Keller, "A possibilistic approach to clustering", IEEE Transactions on Fuzzy Systems, vol. 1, no. 2, pp. 98-110, 1993.
[7] http://cs.joensuu.fi/sipu/datasets/

Code Description

The key codes are described below, check out (https://github.com/waylongo/Gans-for-anomaly-detection) for more information.

- define generator G class

```
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            nn.Linear(random_neuron, 128),
            nn.ReLU(True),
            nn.Linear(128, 128),
            nn.ReLU(True),
            nn.Linear(128, components)
        )
    def forward(self, input):
        return self.main(input)
```

- define discriminator D class

```
class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            nn.Linear(components, 128),
            nn.ReLU(True),
            nn.Linear(128, 128),
            nn.ReLU(True),
            nn.Linear(128, 1),
            nn.Sigmoid()
        )
    def forward(self, input):
        return self.main(input)
```

- main training steps for ensemble GANs

```
for iter in range(G_D_num):

    D = Discriminator(ngpu).to(device)
    G = Generator(ngpu).to(device)
```

```
    # optimization
    opt_D = torch.optim.Adam(D.parameters(), lr=LR_D)
    opt_G = torch.optim.Adam(G.parameters(), lr=LR_G)

    for step in tqdm(range(10000)):
        for _ in range(critic_num):
            # random samples of real data
            idx = np.random.choice(len(data), BATCH_SIZE)
            selected_real = data[idx].float()

            # random noises
            G_noise = torch.randn(BATCH_SIZE, random_neuron).cuda()
            G_data = G(G_noise)

            prob_real = D(selected_real)  # D try to increase this prob
            prob_fake = D(G_data)  # D try to decrease this prob

            D_loss = -torch.mean(torch.log(prob_real + 1e-9) + torch.log(1. - prob_fake + 1e-9))
            G_loss = torch.mean(torch.log(1. - prob_fake + 1e-9))

            opt_D.zero_grad()
            D_loss.backward(retain_graph=True)  # reusing computational graph
            opt_D.step()

        opt_G.zero_grad()
        G_loss.backward()
        opt_G.step()
```

- evaluation (precision, recall and F1 score)

```
test_score = 0.0
for iter in range(G_D_num):
    prediction = D_group['D' + str(iter)](test.float()).cpu().detach().numpy()
    test_score = test_score + prediction / G_D_num

alpha = 0.5 * np.max(z_avg)
for i in range(test_score.shape[0]):
    if test_score[i] > alpha:
        test_score[i] = 0
    else:
        test_score[i] = 1

precision = average_precision_score(label, test_score)
recall = recall_score(label, test_score)
F1 = 2 * (precision * recall) / (precision + recall)
```