1 Assigment 4: Understand anomaly score in GANs project : case study : TadGANs

(T)ime series (a)nomaly (d)etection GANs is stand for TadGANs. The imput model is time series with only two fields in excel file. datetime and values, the output so called anomaly score. This score is related with market crash indication in our research. The aim of TadGANs is to generate anomaly score from closed price and time , to produce anomaly score and use for LSTM or SVM or conGANs model to predict the anomaly score or generate the synthesis signal for anomaly score. Therefore the first step , we should knew about how to generate anomaly score and how to forecast the anomaly score. There exists many program to generate anomaly score, i.e. f-anoGANs, anoGANs, tanoGANs, TadGANs. We choose taDGANs for the first model.

`https://github.com/arunppsg/TadGAN`

The manual of tadGANs are the following and you can find in the link above. This is a Python3 / Pytorch implementation of TadGAN paper. The associated blog explaining the architecture details can be found here.

Data:

The TadGAN architecture can be used for detecting anomalies in time series data.

Pretrained Model:

The trained model is saved in the Model directory. The training is incomplete and the model has to be retrained for other datasets.

Architecture:

The model implements an encoder and decoder as generator and two critics as discriminators as described in the paper. The loss function is wasserstein loss with gradient penalty.

Usage:

Format of the dataset - The dataset should have a column name as signal containing the signals and a column with name anomaly containing the true labels (used during validation).

Delete the contents of the directory Model.

Change the file name exchange-2_cpc_results.csv in main.py to the name of your dataset.

Note:

This is an independent implementation and I am not related to the authors of the paper.

TadGAN: anomaly score generation:source code

For example, you should be able to download the program from github and modified the program to run with test time series for generate anomaly score before doing project GANs. In the interview of this jobs, I was test with Napat, he took 2 days to modified the code below and successful to running the TadGANs to generate the below image and signal for anomaly score. I was test the other but he fail to do it. So it is not simple to do that. For whom I did not test in the interview should ask Napat on how to do that. The first step, install visual stuio code and the second steps down load his program that can run in jupyter book to test running with input excel file. I will tell you steps by steps on how to do in this training. We want to modified the code below to run on matlab with python on matlab later.

```
import os
import logging
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
from torch.utils.data import Dataset, DataLoader
import numpy as np
from scipy import stats

import torch

def test(test_loader, encoder, decoder, critic_x):
    reconstruction_error = list()
    critic_score = list()
    y_true = list()

    for batch, sample in enumerate(test_loader):
        reconstructed_signal = decoder(encoder(sample['signal']))
        reconstructed_signal = torch.squeeze(reconstructed_signal)

        for i in range(0, 64):
```

```
            x_ = reconstructed_signal[i].detach().numpy()
            x = sample['signal'][i].numpy()
            # y_true.append(int(sample['anomaly'][i].detach()))
            reconstruction_error.append(dtw_reconstruction_error(x, x_))
        critic_score.extend(torch.squeeze(critic_x(sample['signal'])).detach().numpy())

    reconstruction_error = stats.zscore(reconstruction_error)
    critic_score = stats.zscore(critic_score)
    anomaly_score = reconstruction_error * critic_score

    return anomaly_score

    # y_predict = detect_anomaly(anomaly_score)
    # y_predict = prune_false_positive(y_predict, anomaly_score, change_threshold=0.1)
    # find_scores(y_true, y_predict)

#Other error metrics - point wise difference, Area difference.
def dtw_reconstruction_error(x, x_):
    n, m = x.shape[0], x_.shape[0]
    dtw_matrix = np.zeros((n+1, m+1))
    for i in range(n+1):
        for j in range(m+1):
            dtw_matrix[i, j] = np.inf
    dtw_matrix[0, 0] = 0

    for i in range(1, n+1):
        for j in range(1, m+1):
            cost = abs(x[i-1] - x_[j-1])
            # take last min from a square box
            last_min = np.min([dtw_matrix[i-1, j], dtw_matrix[i, j-1], dtw_matrix[i-1, j-1]]
            dtw_matrix[i, j] = cost + last_min
    return dtw_matrix[n][m]

def unroll_signal(x):
    x = np.array(x).reshape(100)
    return np.median(x)

def prune_false_positive(is_anomaly, anomaly_score, change_threshold):
    #The model might detect a high number of false positives.
    #In such a scenario, pruning of the false positive is suggested.
    #Method used is as described in the Section 5, part D Identifying Anomalous
    #Sequence, sub-part - Mitigating False positives
    #TODO code optimization
    seq_details = []
    delete_sequence = 0
    start_position = 0
```

```
    end_position = 0
    max_seq_element = anomaly_score[0]
    for i in range(1, len(is_anomaly)):
        if i+1 == len(is_anomaly):
            seq_details.append([start_position, i, max_seq_element, delete_sequence])
        elif is_anomaly[i] == 1 and is_anomaly[i+1] == 0:
            end_position = i
            seq_details.append([start_position, end_position, max_seq_element, delete_sequer
        elif is_anomaly[i] == 1 and is_anomaly[i-1] == 0:
            start_position = i
            max_seq_element = anomaly_score[i]
        if is_anomaly[i] == 1 and is_anomaly[i-1] == 1 and anomaly_score[i] > max_seq_elemer
            max_seq_element = anomaly_score[i]

    max_elements = list()
    for i in range(0, len(seq_details)):
        max_elements.append(seq_details[i][2])

    max_elements.sort(reverse=True)
    max_elements = np.array(max_elements)
    change_percent = abs(max_elements[1:] - max_elements[:-1]) / max_elements[1:]

    #Appending 0 for the 1 st element which is not change percent
    delete_seq = np.append(np.array([0]), change_percent < change_threshold)

    #Mapping max element and seq details
    for i, max_elt in enumerate(max_elements):
        for j in range(0, len(seq_details)):
            if seq_details[j][2] == max_elt:
                seq_details[j][3] = delete_seq[i]

    for seq in seq_details:
        if seq[3] == 1: #Delete sequence
            is_anomaly[seq[0]:seq[1]+1] = [0] * (seq[1] - seq[0] + 1)

    return is_anomaly

def detect_anomaly(anomaly_score):
    window_size = len(anomaly_score) // 3
    step_size = len(anomaly_score) // (3 * 10)

    is_anomaly = np.zeros(len(anomaly_score))

    for i in range(0, len(anomaly_score) - window_size, step_size):
        window_elts = anomaly_score[i:i+window_size]
        window_mean = np.mean(window_elts)
```

```
        window_std = np.std(window_elts)

        for j, elt in enumerate(window_elts):
            if (window_mean - 3 * window_std) < elt < (window_mean + 3 * window_std):
                is_anomaly[i + j] = 0
            else:
                is_anomaly[i + j] = 1

    return is_anomaly

def find_scores(y_true, y_predict):
    tp = tn = fp = fn = 0

    for i in range(0, len(y_true)):
        if y_true[i] == 1 and y_predict[i] == 1:
            tp += 1
        elif y_true[i] == 1 and y_predict[i] == 0:
            fn += 1
        elif y_true[i] == 0 and y_predict[i] == 0:
            tn += 1
        elif y_true[i] == 0 and y_predict[i] == 1:
            fp += 1

    print ('Accuracy {:.2f}'.format((tp + tn)/(len(y_true))))
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    print ('Precision {:.2f}'.format(precision))
    print ('Recall {:.2f}'.format(recall))
    print ('F1 Score {:.2f}'.format(2 * precision * recall / (precision + recall)))

#!/usr/bin/env python
# coding: utf-8

import os
import logging
import numpy as np

import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable
from torch.utils.data import Dataset, DataLoader

class Encoder(nn.Module):

    def __init__(self, encoder_path, signal_shape=100):
```

```
        super(Encoder, self).__init__()
        self.signal_shape = signal_shape
        self.lstm = nn.LSTM(input_size=self.signal_shape, hidden_size=20, num_layers=1, bidi
        self.dense = nn.Linear(in_features=40, out_features=20)
        self.encoder_path = encoder_path

    def forward(self, x):
        x = x.view(1, 64, self.signal_shape).float()
        x, (hn, cn) = self.lstm(x)
        x = self.dense(x)
        return (x)

class Decoder(nn.Module):
    def __init__(self, decoder_path, signal_shape=100):
        super(Decoder, self).__init__()
        self.signal_shape = signal_shape
        self.lstm = nn.LSTM(input_size=20, hidden_size=64, num_layers=2, bidirectional=True)
        self.dense = nn.Linear(in_features=128, out_features=self.signal_shape)
        self.decoder_path = decoder_path

    def forward(self, x):
        x, (hn, cn) = self.lstm(x)
        x = self.dense(x)
        return (x)

class CriticX(nn.Module):
    def __init__(self, critic_x_path, signal_shape=100):
        super(CriticX, self).__init__()
        self.signal_shape = signal_shape
        self.dense1 = nn.Linear(in_features=self.signal_shape, out_features=20)
        self.dense2 = nn.Linear(in_features=20, out_features=1)
        self.critic_x_path = critic_x_path

    def forward(self, x):
        x = x.view(1, 64, self.signal_shape).float()
        x = self.dense1(x)
        x = self.dense2(x)
        return (x)

class CriticZ(nn.Module):
    def __init__(self, critic_z_path):
        super(CriticZ, self).__init__()
        self.dense1 = nn.Linear(in_features=20, out_features=1)
        self.critic_z_path = critic_z_path

    def forward(self, x):
```

```
        x = self.dense1(x)
        return (x)

def unroll_signal(self, x):
    x = np.array(x).reshape(100)
    return np.median(x)

# def test(self):
#     """
#     Returns a dataframe with original value, reconstructed value, reconstruction error, cr
#     """
#     df = self.test_dataset.copy()
#     X_ = list()

#     RE = list()  #Reconstruction error
#     CS = list()  #Critic score

#     for i in range(0, df.shape[0]):
#         x = df.rolled_signal[i]
#         x = tf.reshape(x, (1, 100, 1))
#         z = encoder(x)
#         z = tf.expand_dims(z, axis=2)
#         x_ = decoder(z)

#         re = dtw_reconstruction_error(tf.squeeze(x_).numpy(), tf.squeeze(x).numpy()) #reco
#         cs = critic_x(x)
#         cs = tf.squeeze(cs).numpy()
#         RE.append(re)
#         CS.append(cs)

#         x_ = unroll_signal(x_)

#         X_.append(x_)

#     df['generated_signals'] = X_

#     return df

logging.basicConfig(filename='train.log', level=logging.DEBUG)

class SignalDataset(Dataset):
    def __init__(self, path):
        self.signal_df = pd.read_csv(path)
        self.signal_columns = self.make_signal_list()
        self.make_rolling_signals()
```

```
    def make_signal_list(self):
        signal_list = list()
        for i in range(-50, 50):
            signal_list.append('signal'+str(i))
        return signal_list

    def make_rolling_signals(self):
        for i in range(-50, 50):
            self.signal_df['signal'+str(i)] = self.signal_df['signal'].shift(i)
        self.signal_df = self.signal_df.dropna()
        self.signal_df = self.signal_df.reset_index(drop=True)

    def __len__(self):
        return len(self.signal_df)

    def __getitem__(self, idx):
        row = self.signal_df.loc[idx]
        x = row[self.signal_columns].values.astype(float)
        x = torch.from_numpy(x)
        return {'signal': x}

def critic_x_iteration(sample):
    optim_cx.zero_grad()

    x = sample['signal'].view(1, batch_size, signal_shape)
    valid_x = critic_x(x)
    valid_x = torch.squeeze(valid_x)
    critic_score_valid_x = torch.mean(torch.ones(valid_x.shape) * valid_x) #Wasserstein Loss

    #The sampled z are the anomalous points - points deviating from actual distribution of z
    z = torch.empty(1, batch_size, latent_space_dim).uniform_(0, 1)
    x_ = decoder(z)
    fake_x = critic_x(x_)
    fake_x = torch.squeeze(fake_x)
    critic_score_fake_x = torch.mean(torch.ones(fake_x.shape) * fake_x)  #Wasserstein Loss

    alpha = torch.rand(x.shape)
    ix = Variable(alpha * x + (1 - alpha) * x_) #Random Weighted Average
    ix.requires_grad_(True)
    v_ix = critic_x(ix)
    v_ix.mean().backward()
    gradients = ix.grad
    #Gradient Penalty Loss
    gp_loss = torch.sqrt(torch.sum(torch.square(gradients).view(-1)))

    #Critic has to maximize Cx(Valid X) - Cx(Fake X).
```

```
    #Maximizing the above is same as minimizing the negative.
    wl = critic_score_fake_x - critic_score_valid_x
    loss = wl + gp_loss
    loss.backward()
    optim_cx.step()

    return loss

def critic_z_iteration(sample):
    optim_cz.zero_grad()

    x = sample['signal'].view(1, batch_size, signal_shape)
    z = encoder(x)
    valid_z = critic_z(z)
    valid_z = torch.squeeze(valid_z)
    critic_score_valid_z = torch.mean(torch.ones(valid_z.shape) * valid_z)

    z_ = torch.empty(1, batch_size, latent_space_dim).uniform_(0, 1)
    fake_z = critic_z(z_)
    fake_z = torch.squeeze(fake_z)
    critic_score_fake_z = torch.mean(torch.ones(fake_z.shape) * fake_z) #Wasserstein Loss

    wl = critic_score_fake_z - critic_score_valid_z

    alpha = torch.rand(z.shape)
    iz = Variable(alpha * z + (1 - alpha) * z_) #Random Weighted Average
    iz.requires_grad_(True)
    v_iz = critic_z(iz)
    v_iz.mean().backward()
    gradients = iz.grad
    gp_loss = torch.sqrt(torch.sum(torch.square(gradients).view(-1)))

    loss = wl + gp_loss
    loss.backward()
    optim_cz.step()

    return loss

def encoder_iteration(sample):
    optim_enc.zero_grad()

    x = sample['signal'].view(1, batch_size, signal_shape)
    valid_x = critic_x(x)
    valid_x = torch.squeeze(valid_x)
    critic_score_valid_x = torch.mean(torch.ones(valid_x.shape) * valid_x) #Wasserstein Loss
```

```
    z = torch.empty(1, batch_size, latent_space_dim).uniform_(0, 1)
    x_ = decoder(z)
    fake_x = critic_x(x_)
    fake_x = torch.squeeze(fake_x)
    critic_score_fake_x = torch.mean(torch.ones(fake_x.shape) * fake_x)

    enc_z = encoder(x)
    gen_x = decoder(enc_z)

    mse = mse_loss(x.float(), gen_x.float())
    loss_enc = mse + critic_score_valid_x - critic_score_fake_x
    loss_enc.backward(retain_graph=True)
    optim_enc.step()

    return loss_enc

def decoder_iteration(sample):
    optim_dec.zero_grad()

    x = sample['signal'].view(1, batch_size, signal_shape)
    z = encoder(x)
    valid_z = critic_z(z)
    valid_z = torch.squeeze(valid_z)
    critic_score_valid_z = torch.mean(torch.ones(valid_z.shape) * valid_z)

    z_ = torch.empty(1, batch_size, latent_space_dim).uniform_(0, 1)
    fake_z = critic_z(z_)
    fake_z = torch.squeeze(fake_z)
    critic_score_fake_z = torch.mean(torch.ones(fake_z.shape) * fake_z)

    enc_z = encoder(x)
    gen_x = decoder(enc_z)

    mse = mse_loss(x.float(), gen_x.float())
    loss_dec = mse + critic_score_valid_z - critic_score_fake_z
    loss_dec.backward(retain_graph=True)
    optim_dec.step()

    return loss_dec


def train(n_epochs=2000):
    logging.debug('Starting training')
    cx_epoch_loss = list()
    cz_epoch_loss = list()
    encoder_epoch_loss = list()
```

```
    decoder_epoch_loss = list()

    for epoch in range(n_epochs):
        logging.debug('Epoch {}'.format(epoch))
        n_critics = 5

        cx_nc_loss = list()
        cz_nc_loss = list()

        for i in range(n_critics):
            cx_loss = list()
            cz_loss = list()

            for batch, sample in enumerate(train_loader):
                loss = critic_x_iteration(sample)
                cx_loss.append(loss)

                loss = critic_z_iteration(sample)
                cz_loss.append(loss)

            cx_nc_loss.append(torch.mean(torch.tensor(cx_loss)))
            cz_nc_loss.append(torch.mean(torch.tensor(cz_loss)))

        logging.debug('Critic training done in epoch {}'.format(epoch))
        encoder_loss = list()
        decoder_loss = list()

        for batch, sample in enumerate(train_loader):
            enc_loss = encoder_iteration(sample)
            dec_loss = decoder_iteration(sample)
            encoder_loss.append(enc_loss)
            decoder_loss.append(dec_loss)

        cx_epoch_loss.append(torch.mean(torch.tensor(cx_nc_loss)))
        cz_epoch_loss.append(torch.mean(torch.tensor(cz_nc_loss)))
        encoder_epoch_loss.append(torch.mean(torch.tensor(encoder_loss)))
        decoder_epoch_loss.append(torch.mean(torch.tensor(decoder_loss)))
        logging.debug('Encoder decoder training done in epoch {}'.format(epoch))
        logging.debug('critic x loss {:.3f} critic z loss {:.3f} \nencoder loss {:.3f}
decoder loss {:.3f}\n'.format(cx_epoch_loss[-1],
cz_epoch_loss[-1], encoder_epoch_loss[-1], decoder_epoch_loss[-1]))

        if epoch % 10 == 0:
            torch.save(encoder.state_dict(), encoder.encoder_path)
            torch.save(decoder.state_dict(), decoder.decoder_path)
            torch.save(critic_x.state_dict(), critic_x.critic_x_path)
```

```
            torch.save(critic_z.state_dict(), critic_z.critic_z_path)
dataset = pd.read_csv('test.csv')
#Splitting intro train and test
#TODO could be done in a more pythonic way
train_len = int(0.7 * dataset.shape[0])
dataset[0:train_len].to_csv('train_dataset.csv', index=False)
dataset[train_len:].to_csv('test_dataset.csv', index=False)

train_dataset = SignalDataset(path='train_dataset.csv')
test_dataset = SignalDataset(path='test_dataset.csv')
batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, drop_last=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, drop_last=True)

logging.info('Number of train datapoints is {}'.format(len(train_dataset)))
logging.info('Number of samples in train dataset {}'.format(len(train_dataset)))

lr = 1e-6

signal_shape = 100
latent_space_dim = 20
encoder_path = 'models/encoder.pt'
decoder_path = 'models/decoder.pt'
critic_x_path = 'models/critic_x.pt'
critic_z_path = 'models/critic_z.pt'

encoder = Encoder(encoder_path, signal_shape)
decoder = Decoder(decoder_path, signal_shape)
critic_x = CriticX(critic_x_path, signal_shape)
critic_z = CriticZ(critic_z_path)

mse_loss = torch.nn.MSELoss()

optim_enc = optim.Adam(encoder.parameters(), lr=lr, betas=(0.5, 0.999))
optim_dec = optim.Adam(decoder.parameters(), lr=lr, betas=(0.5, 0.999))
optim_cx = optim.Adam(critic_x.parameters(), lr=lr, betas=(0.5, 0.999))
optim_cz = optim.Adam(critic_z.parameters(), lr=lr, betas=(0.5, 0.999))

train(n_epochs=1)

test(test_loader, encoder, decoder, critic_x)

input_dataset = SignalDataset(path='dataset_input.csv')
print(f'Dataset size: {len(input_dataset)}')
data_newinput = DataLoader(input_dataset, batch_size=64, drop_last=True)
print(f'Dataset size: {len(data_newinput)}')
```

```
train(n_epochs=5)

anomaly_score = test(data_newinput, encoder, decoder, critic_x)
import pandas as pd

def load_dataset(file_path):

    df = pd.read_csv(file_path)


    return df

file_path = 'dataset_input.csv'
df = load_dataset(file_path)

anomaly_score
df
if len(anomaly_score) == len(df):
    df['Anomaly Score'] = anomaly_score
else:
    print("Length of anomaly_score does not match length of DataFrame")
len(anomaly_score)
num_rows = len(df)
if len(anomaly_score) < num_rows:

    anomaly_score = np.concatenate([anomaly_score, np.zeros(num_rows - len(anomaly_score))])
elif len(anomaly_score) > num_rows:

    anomaly_score = anomaly_score[:num_rows]

df['Anomaly Score'] = anomaly_score
df
plt.figure(figsize=(12, 6))

plt.plot(df.index, df['signal'], label='Signal', color='blue', alpha=0.7)


plt.plot(df.index, df['Anomaly Score'], label='Anomaly Score', color='red', alpha=0.7)

plt.title('Signal and Anomaly Score')
plt.xlabel('Index')
plt.ylabel('Value')
plt.legend()
plt.grid(True)

plt.show()
```

```
df_subset = df.head(100)


plt.figure(figsize=(12, 6))


plt.plot(df_subset.index, df_subset['signal'], label='Signal', color='blue', alpha=0.7)

plt.plot(df_subset.index, df_subset['Anomaly Score'], label='Anomaly Score', color='red', al

plt.title('Signal and Anomaly Score (First 100 Rows)')
plt.xlabel('Index')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
pd.set_option('display.max_rows', 200)
df.head(100)
```

Input excel file: dataset_input.csv and output excel file:train.csv, test.csv

Input have 2 fields to test time series with 1,2,3,4,100,200,1,0,1,0,1,.....

```
timestamp signal
0 1/7/11 00:00 1
1 1/7/11 01:00 2
2 1/7/11 02:00 3
3 1/7/11 03:00 4
4 1/7/11 04:00 5
... ... ...
994 11/8/11 13:00 1
995 11/8/11 14:00 0
996 11/8/11 15:00 1
997 11/8/11 16:00 0
998 11/8/11 17:00 1
999 rows × 2 columns
```

The out put of running is anomaly score:

```
timestamp signal Anomaly Score
0 1/7/11 00:00 1 -27.358179
1 1/7/11 01:00 2 -23.992388
2 1/7/11 02:00 3 8.437466
```

```
3 1/7/11 03:00 4 38.998158
4 1/7/11 04:00 5 35.680802
5 1/7/11 05:00 6 -18.610216
6 1/7/11 06:00 7 -30.153104
7 1/7/11 07:00 8 -28.080890
8 1/7/11 08:00 10 -6.219011
9 1/7/11 09:00 100 -53.584678
10 1/7/11 10:00 200 -34.536885
11 1/7/11 11:00 500 -28.719962
12 1/7/11 12:00 1 16.166623
13 1/7/11 13:00 2 11.607382
14 1/7/11 14:00 3 -1.745348
15 1/7/11 15:00 4 -8.293290
16 1/7/11 16:00 5 -6.088623
17 1/7/11 17:00 6 -3.330445
18 1/7/11 18:00 7 -2.622109
19 1/7/11 19:00 8 11.373294
20 1/7/11 20:00 9 -2.101552
21 1/7/11 21:00 10 -2.476517
22 1/7/11 22:00 1 -2.354532
23 1/7/11 23:00 2 16.527498
24 2/7/11 00:00 3 22.855227
25 2/7/11 01:00 4 -5.972269
26 2/7/11 02:00 5 -17.820204
27 2/7/11 03:00 6 -8.224134
28 2/7/11 04:00 7 -1.708896
29 2/7/11 05:00 8 1.516771
30 2/7/11 06:00 9 19.838441
31 2/7/11 07:00 10 20.614057
32 2/7/11 08:00 1 13.766756
33 2/7/11 09:00 2 -8.376410
34 2/7/11 10:00 3 10.000592
35 2/7/11 11:00 4 2.275739
36 2/7/11 12:00 5 5.526800
37 2/7/11 13:00 6 -3.859873
38 2/7/11 14:00 7 -13.388280
39 2/7/11 15:00 8 -4.831089
40 2/7/11 16:00 9 4.877447
41 2/7/11 17:00 10 6.388901
42 2/7/11 18:00 1 0.405545
43 2/7/11 19:00 2 -8.552155
44 2/7/11 20:00 3 -7.121125
45 2/7/11 21:00 4 5.036886
46 2/7/11 22:00 5 14.351978
47 2/7/11 23:00 6 13.108573
48 3/7/11 00:00 7 15.689000
```

```
49 3/7/11 01:00 8 -6.553532
50 3/7/11 02:00 9 -9.967559
51 3/7/11 03:00 10 -12.371233
52 3/7/11 04:00 10 -8.635651
53 3/7/11 05:00 100 -17.303339
54 3/7/11 06:00 200 -15.298918
55 3/7/11 07:00 500 -7.217293
56 3/7/11 08:00 10 -0.000272
57 3/7/11 09:00 1 -0.005698
58 3/7/11 10:00 2 -0.000175
59 3/7/11 11:00 3 0.001119
60 3/7/11 12:00 4 0.011353
61 3/7/11 13:00 5 0.008347
62 3/7/11 14:00 6 0.012928
63 3/7/11 15:00 7 0.008575
64 3/7/11 16:00 8 0.021170
65 3/7/11 17:00 9 0.020046
66 3/7/11 18:00 10 0.024739
67 3/7/11 19:00 1 0.001715
68 3/7/11 20:00 2 0.014684
69 3/7/11 21:00 3 0.004528
70 3/7/11 22:00 4 0.017133
71 3/7/11 23:00 5 -0.006223
72 4/7/11 00:00 1 0.001235
73 4/7/11 01:00 0 -0.017330
74 4/7/11 02:00 1 0.001235
75 4/7/11 03:00 0 -0.017330
76 4/7/11 04:00 1 0.001235
77 4/7/11 05:00 0 -0.017330
78 4/7/11 06:00 1 0.001235
79 4/7/11 07:00 0 -0.017330
80 4/7/11 08:00 1 0.001235
81 4/7/11 09:00 0 -0.017330
82 4/7/11 10:00 1 0.001235
83 4/7/11 11:00 0 -0.017330
84 4/7/11 12:00 1 0.001235
85 4/7/11 13:00 0 -0.017330
86 4/7/11 14:00 1 0.001235
87 4/7/11 15:00 0 -0.017330
88 4/7/11 16:00 1 0.001235
89 4/7/11 17:00 0 -0.017330
90 4/7/11 18:00 1 0.001235
91 4/7/11 19:00 0 -0.017330
92 4/7/11 20:00 1 0.001235
93 4/7/11 21:00 0 -0.017330
94 4/7/11 22:00 1 0.001235
```

```
95 4/7/11 23:00 0 -0.017330
96 5/7/11 00:00 1 0.001235
97 5/7/11 01:00 0 -0.017330
98 5/7/11 02:00 1 0.001235
99 5/7/11 03:00 0 -0.017330
```