Timeseries anomaly detection using an Autoencoder

AM202216702_SA JIM SOE MOE

TABLE OF CONTENTS

Introduction

Anomaly is something that deviates from what is standard, normal, or expected.

Anomaly detection (aka outlier analysis) is a step in data mining that identifies data points, events, and/or observations that deviate from a dataset's normal behavior.

Supervised Anomaly Detection

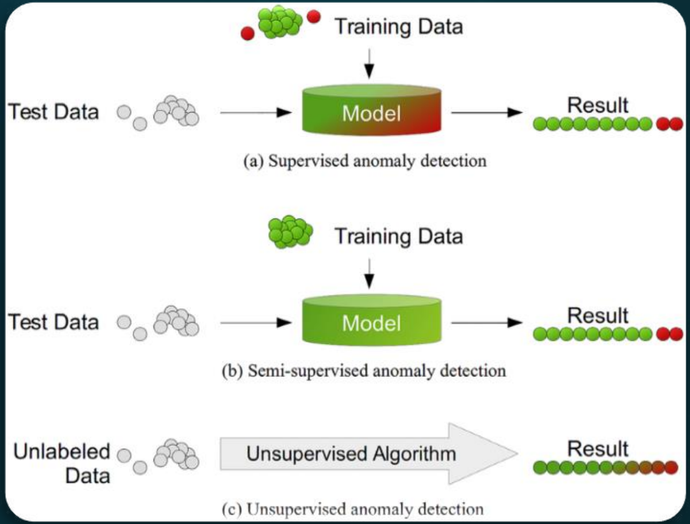
- Uses a fully labeled dataset for training.

Semi-supervised Anomaly Detection

- Uses an anomaly-free training dataset. Afterwards, deviations in the test data from that normal model are used to detect anomalies.

Unsupervised Anomaly Detection

- Uses only intrinsic information of the data in order to detect instances deviating from the majority of the data.



Dataset Overview

DATA SET

- The Numenta Anomaly Benchmark (NAB)
- Artificial timeseries data containing labeled anomalous periods of behavior.
- Data are ordered, timestamped, single-valued metrics.
- The time period is from 1 - 14 April of 2014.

Source

- https://raw.githubusercontent.com/numenta/NAB/master/data/

Without anomaly

timestamp	value
4/1/2014 0:00	18.32491854
4/1/2014 0:05	21.97032718
4/1/2014 0:10	18.62480603
4/1/2014 0:15	21.95368398
4/1/2014 0:20	21.90911973
4/1/2014 0:25	21.17527242
4/1/2014 0:30	20.63769185
4/1/2014 0:35	20.3112282
4/1/2014 0:40	21.46440618
4/1/2014 0:45	19.15775809
4/1/2014 0:50	19.87072485
4/1/2014 0:55	20.47755988
4/1/2014 1:00	19.6447619
4/1/2014 1:05	19.70994582
4/1/2014 1:10	19.32113867
4/1/2014 1:15	20.25692727
4/1/2014 1:20	21.40229811
4/1/2014 1:25	18.80611351
4/1/2014 1:30	21.73773216
4/1/2014 1:35	20.75635062
4/1/2014 1:40	21.29309285
4/1/2014 1:45	20.22476277
4/1/2014 1:50	21.11806681
4/1/2014 1:55	18.06480159
4/1/2014 2:00	21.2735217
4/1/2014 2:05	18.16055544
4/1/2014 2:10	21.55965351

With anomaly

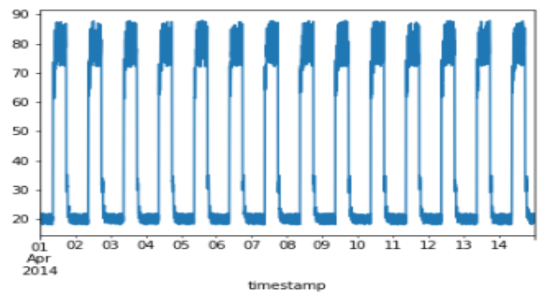
timestamp	value
4/1/2014 0:00	19.7612519
4/1/2014 0:05	20.50083329
4/1/2014 0:10	19.96164144
4/1/2014 0:15	21.49026607
4/1/2014 0:20	20.18773941
4/1/2014 0:25	19.92312567
4/1/2014 0:30	21.69840396
4/1/2014 0:35	20.87875838
4/1/2014 0:40	18.44619963
4/1/2014 0:45	18.71081784
4/1/2014 0:50	21.14849145
4/1/2014 0:55	21.34340528
4/1/2014 1:00	20.18076332
4/1/2014 1:05	20.21782091
4/1/2014 1:10	20.52773185
4/1/2014 1:15	19.7564631
4/1/2014 1:20	20.72079649
4/1/2014 1:25	18.43392503
4/1/2014 1:30	21.84511697
4/1/2014 1:35	21.0006193
4/1/2014 1:40	20.52469682
4/1/2014 1:45	19.26528823
4/1/2014 1:50	18.64382195
4/1/2014 1:55	19.63737186
4/1/2014 2:00	20.64630696
4/1/2014 2:05	20.53483897
4/1/2014 2:10	19.53056462

Visualize the datasets

Timeseries data without anomalies

- We will use the following data for training.
- We will find MAE loss value.
- We will make this value as the threshold for anomaly detection.

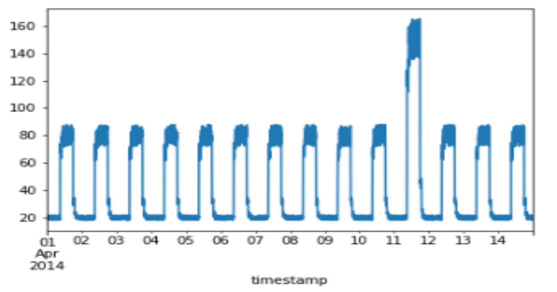
```
# Timeseries data without anomalies

fig, ax = plt.subplots()
df_small_noise.plot(legend=False, ax=ax)
plt.show()
```



Timeseries data with anomalies

- We will use the following data for testing.
- We will see if the sudden jump up in the data is detected as an anomaly.

```
# Timeseries data with anomalies

fig, ax = plt.subplots()
df_daily_jumpsup.plot(legend=False, ax=ax)
plt.show()
```



Prepare Training Data

- Get data values from the training timeseries data file and normalize the value data.
- We have a value for every 5 mins for 14 days.
  - ✓ 24 * 60 / 5 = 288 timesteps per day
  - ✓ 288 * 14 = 4032 data points in total

```
training_mean = df_small_noise.mean()
training_std = df_small_noise.std()
df_training_value = (df_small_noise - training_mean) / training_std
print("Number of training samples:", len(df_training_value))
```

```
Number of training samples: 4032
```

Create Sequences

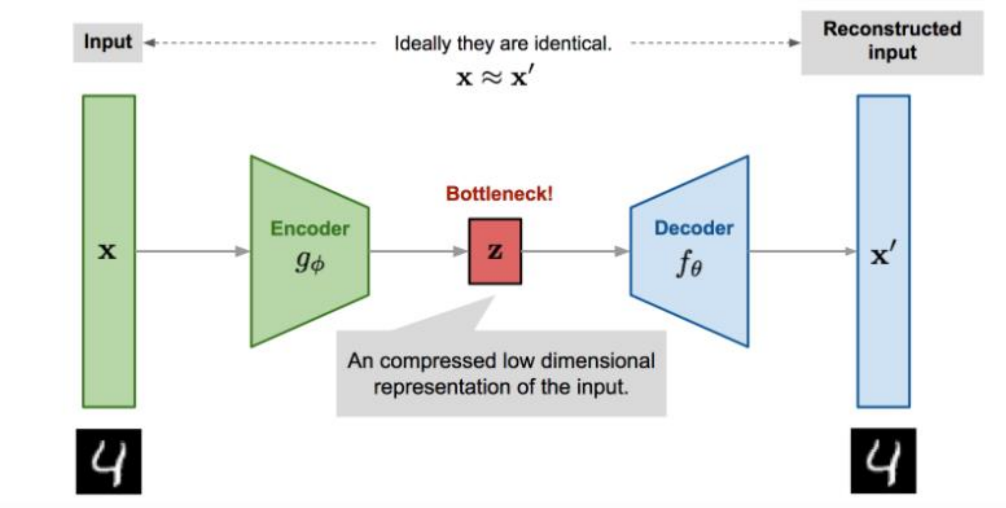
- Create sequences combining TIME_STEPS contiguous data values from the training data.
- TIME_STEPS is set 288 as we want our network to have memory of 288 timesteps which is a day.

```
TIME_STEPS = 288

# Generated training sequences for use in the model.
def create_sequences(values, time_steps=TIME_STEPS):
    output = []
    for i in range(len(values) - time_steps + 1):
        output.append(values[i : (i + time_steps)])
    return np.stack(output)


x_train = create_sequences(df_training_value.values)
print("Training input shape: ", x_train.shape)
```

```
Training input shape:  (3745, 288, 1)
```

Autoencoder

- An autoencoder is a special type of neural network that is trained to copy its input to its output.
- It learns to compress the data while minimizing the reconstruction error.
- It is considered an unsupervised learning technique since it does not require a separate label value to train.
- In practice, the autoencoder is composed of two phases:
  - Encoder: reduces the dimensions of the input data X
  - Decoder: trained to obtain the output data similar to the original space



Building Model

- The model will take input of shape (batch_size, sequence_length, num_features) and return output of the same shape.
- In this case, sequence_length is 288 and num_features is 1.

- Dropout is used for regularization.
- Conv1D means each channel in the input and filter is 1 dimensional.
- Conv1DTranspose is a transposed convolution layer (sometimes called Deconvolution).

```
model = keras.Sequential(
    [
        layers.Input(shape=(x_train.shape[1], x_train.shape[2])),
        layers.Conv1D(filters=32, kernel_size=7, padding="same", strides=2, activation="relu"),
        layers.Dropout(rate=0.2),
        layers.Conv1D(filters=16, kernel_size=7, padding="same", strides=2, activation="relu"),
        layers.Conv1DTranspose(filters=16, kernel_size=7, padding="same", strides=2, activation="relu"),
        layers.Dropout(rate=0.2),
        layers.Conv1DTranspose(filters=32, kernel_size=7, padding="same", strides=2, activation="relu"),
        layers.Conv1DTranspose(filters=1, kernel_size=7, padding="same"),
    ]
)
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss="mse")
model.summary()
```

Model: "sequential"

	Layer (type)	Output Shape	Param #
Encoder	conv1d (Conv1D)	(None, 144, 32)	256
	dropout (Dropout)	(None, 144, 32)	0
	conv1d_1 (Conv1D)	(None, 72, 16)	3600
Decoder	conv1d_transpose (Conv1DTran	(None, 144, 16)	1808
	dropout_1 (Dropout)	(None, 144, 16)	0
	conv1d_transpose_1 (Conv1DTr	(None, 288, 32)	3616
	conv1d_transpose_2 (Conv1DTr	(None, 288, 1)	225

Total params: 9,505
Trainable params: 9,505
Non-trainable params: 0

Training Model

Epochs

- Epoch is when an entire dataset is passed forward and backward through the neural network only once.

Batch_size

- Since one epoch is too big to feed to the computer at once we divide it in several smaller batches.

Validation_split

- Fraction of the training data to be used as validation data.

Callback

- An object that performs actions at various stages of training

Monitor

- Quantity to be monitored.

Patience

- Number of epochs with no improvement after which training will be stopped.

Mode

- In min mode, training will stop when the quantity monitored has stopped decreasing

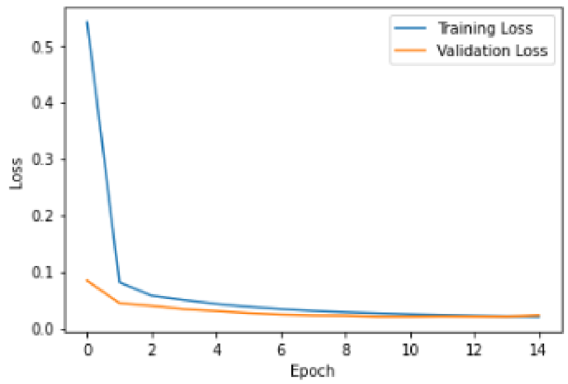
```python
history = model.fit(
    x_train,
    x_train,
    epochs=50,
    batch_size=128,
    validation_split=0.1,
    callbacks=[
        keras.callbacks.EarlyStopping(monitor="val_loss", patience=5, mode="min")
    ],
)
```

```
Epoch 1/50
2022-05-03 05:14:46.657003: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:18
27/27 [==============================] - 2s 47ms/step - loss: 0.4893 - val_loss: 0.0553
Epoch 2/50
27/27 [==============================] - 1s 36ms/step - loss: 0.0770 - val_loss: 0.0460
Epoch 3/50
27/27 [==============================] - 1s 37ms/step - loss: 0.0601 - val_loss: 0.0404
Epoch 4/50
27/27 [==============================] - 1s 39ms/step - loss: 0.0533 - val_loss: 0.0363
Epoch 5/50
27/27 [==============================] - 1s 37ms/step - loss: 0.0474 - val_loss: 0.0328
Epoch 6/50
27/27 [==============================] - 1s 37ms/step - loss: 0.0421 - val_loss: 0.0293
Epoch 7/50
27/27 [==============================] - 1s 39ms/step - loss: 0.0378 - val_loss: 0.0277
Epoch 8/50
27/27 [==============================] - 1s 37ms/step - loss: 0.0343 - val_loss: 0.0260
Epoch 9/50
27/27 [==============================] - 1s 36ms/step - loss: 0.0314 - val_loss: 0.0247
Epoch 10/50
27/27 [==============================] - 1s 37ms/step - loss: 0.0290 - val_loss: 0.0231
Epoch 11/50
27/27 [==============================] - 1s 37ms/step - loss: 0.0273 - val_loss: 0.0218
Epoch 12/50
27/27 [==============================] - 1s 37ms/step - loss: 0.0258 - val_loss: 0.0224
```

Plot Training and Validation loss

- The training loss indicates how well the model is fitting the training data.
- The validation loss indicates how well the model fits new data.

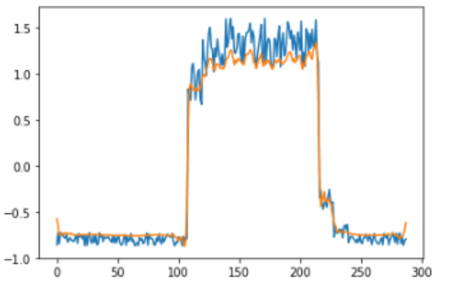
```python
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



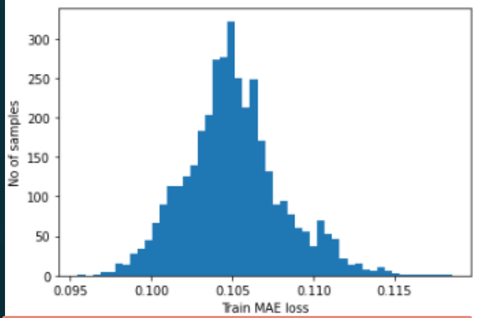
Detecting anomalies

- We will detect anomalies by determining how well our model can reconstruct the input data.
  - ✓ Find MAE loss on training samples.
  - ✓ Find max MAE loss value and make this the threshold for anomaly detection.
  - ✓ If the reconstruction loss for a sample is greater than this threshold value, we will label this sample as an anomaly.

```python
# Checking how the first sequence is learnt
plt.plot(x_train[0])
plt.plot(x_train_pred[0])
plt.show()
```




```python
# Get train MAE loss.
x_train_pred = model.predict(x_train)
train_mae_loss = np.mean(np.abs(x_train_pred - x_train), axis=1)

plt.hist(train_mae_loss, bins=50)
plt.xlabel("Train MAE loss")
plt.ylabel("No of samples")
plt.show()

# Get reconstruction loss threshold.
threshold = np.max(train_mae_loss)
print("Reconstruction error threshold: ", threshold)
```

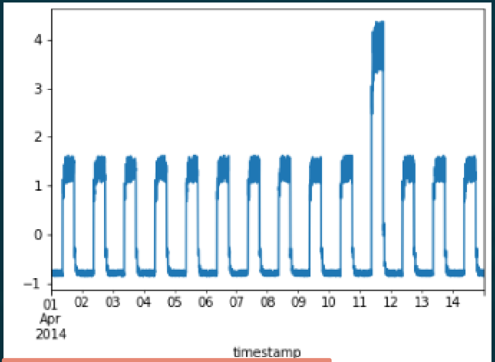



```
Reconstruction error threshold:  0.11853996076347365
```

Prepare test data

- Get test values from the test timeseries data file and normalize the value data.
- Same as training data, we have a value for every 5 mins for 14 days.
- Create sequences from test values.

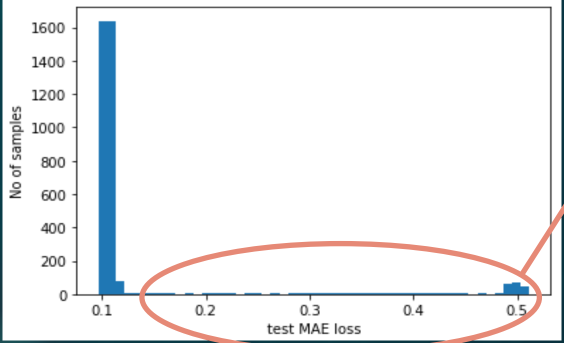
```python
df_test_value = (df_daily_jumpsup - training_mean) / training_std
fig, ax = plt.subplots()
df_test_value.plot(legend=False, ax=ax)
plt.show()

# Create sequences from test values.
x_test = create_sequences(df_test_value.values)
print("Test input shape: ", x_test.shape)
```




```
Test input shape:  (3745, 288, 1)
```

Predict Anomalies on test data using threshold

```python
# Get test MAE loss.
x_test_pred = model.predict(x_test)
test_mae_loss = np.mean(np.abs(x_test_pred - x_test), axis=1)
test_mae_loss = test_mae_loss.reshape((-1))

plt.hist(test_mae_loss, bins=50)
plt.xlabel("test MAE loss")
plt.ylabel("No of samples")
plt.show()
```



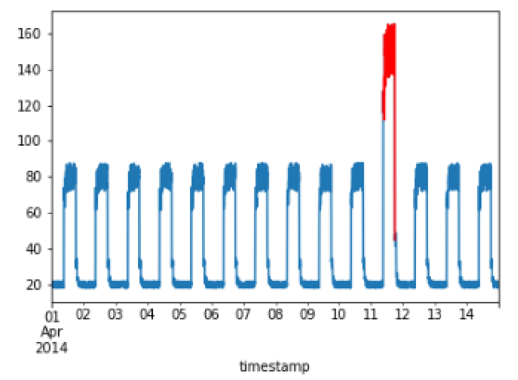

```python
# Detect all the samples which are anomalies.
anomalies = test_mae_loss > threshold
print("Number of anomaly samples: ", np.sum(anomalies))
print("Indices of anomaly samples: ", np.where(anomalies))
```

```
Number of anomaly samples:  410
Indices of anomaly samples:  (array([1651, 1652, 1653, 1654, 1655, 1657, 1658, 1659, 1660, 2517, 2521,
       2522, 2523, 2524, 2697, 2701, 2702, 2703, 2704, 2705, 2706, 2707,
       2708, 2709, 2710, 2711, 2712, 2713, 2714, 2715, 2716, 2717, 2718,
       2719, 2720, 2721, 2722, 2723, 2724, 2725, 2726, 2727, 2728, 2729,
       2730, 2731, 2732, 2733, 2734, 2735, 2736, 2737, 2738, 2739, 2740,
       2741, 2742, 2743, 2744, 2745, 2746, 2747, 2748, 2749, 2750, 2751,
       2752, 2753, 2754, 2755, 2756, 2757, 2758, 2759, 2760, 2761, 2762,
       2763, 2764, 2765, 2766, 2767, 2768, 2769, 2770, 2771, 2772, 2773,
       2774, 2775, 2776, 2777, 2778, 2779, 2780, 2781, 2782, 2783, 2784,
       2785, 2786, 2787, 2788, 2789, 2790, 2791, 2792, 2793, 2794, 2795,
       2796, 2797, 2798, 2799, 2800, 2801, 2802, 2803, 2804, 2805, 2806,
       2807, 2808, 2809, 2810, 2811, 2812, 2813, 2814, 2815, 2816, 2817,
       2818, 2819, 2820, 2821, 2822, 2823, 2824, 2825, 2826, 2827, 2828,
       2829, 2830, 2831, 2832, 2833, 2834, 2835, 2836, 2837, 2838, 2839,
       2840, 2841, 2842, 2843, 2844, 2845, 2846, 2847, 2848, 2849, 2850,
       2851, 2852, 2853, 2854, 2855, 2856, 2857, 2858, 2859, 2860, 2861,
       2862, 2863, 2864, 2865, 2866, 2867, 2868, 2869, 2870, 2871, 2872,
       2873, 2874, 2875, 2876, 2877, 2878, 2879, 2880, 2881, 2882, 2883,
       2884, 2885, 2886, 2887, 2888, 2889, 2890, 2891, 2892, 2893, 2894,
```

Plot anomalies

- We now know the samples of the data which are anomalies.
- With this, we will find the corresponding timestamps from the original test data.
- Overlay the anomalies on the original test data plot..

```python
# data i is an anomaly if samples [(i - timesteps + 1) to (i)] are anomalies
anomalous_data_indices = []
for data_idx in range(TIME_STEPS - 1, len(df_test_value) - TIME_STEPS + 1):
    if np.all(anomalies[data_idx - TIME_STEPS + 1 : data_idx]):
        anomalous_data_indices.append(data_idx)

df_subset = df_daily_jumpsup.iloc[anomalous_data_indices]
fig, ax = plt.subplots()
df_daily_jumpsup.plot(legend=False, ax=ax)
df_subset.plot(legend=False, ax=ax, color="r")
plt.show()
```



Thanks