진리장학금 1차보고서

by Hyeonwoo Yoo

Table of Contents

- 1 진리장학금 1차보고서
- 2 Objective
- 3 About Novelty Detection, Anomaly Detection
  - 3.1 Anomaly detection
  - 3.2 Novelty Detection
- 4 Evaluation Metrics for Anomaly & Novelty Detection
  - 4.1 Metrics for Anomaly & Novelty Detection
  - 4.2 ROC (Receiver Operating Characteristics)
  - 4.3 AUC (Area Under the Curve)
  - 4.4 PRC (Precision-Recall Curve)
  - 4.5 Better metric for class-imbalanced data
- 5 About Dataset
- 6 Exploratory Data Analysis
- 7 Novelty Detection using PCA Reconstruction
  - 7.1 Number of Components = 2
  - 7.2 Number of Components = 3
  - 7.3 Conclusion
- 8 Novelty Detection using VAE Reconsturction Error
  - 8.1 Bottleneck Dimension = 2
  - 8.2 Bottleneck Dimension = 3
  - 8.3 Conclusion

Objective

- Novely Detection / Anomaly Detection on NASA Bearing Dataset (http://data-acoustics.com/measurements/bearing-faults/bearing-4/)
- Approach with two reconstruction methods
  - Principal Component Analysis Reconsturciton
  - Variatioanl Autoencoder Reconstruction

About Novelty Detection, Anomaly Detection

Anomaly detection

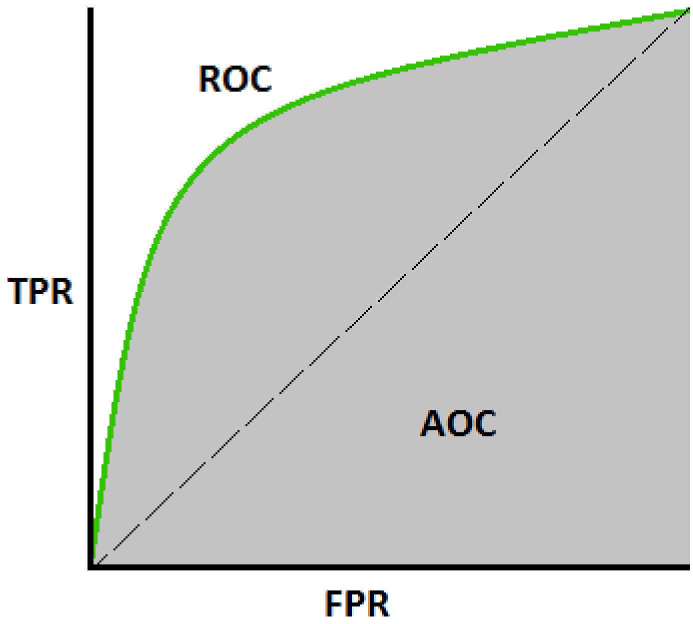
- **Identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data**
- Three methods for anomaly detectoin
  - **Unsupervised** : Unlabeled test data set under the assumption that the majority of the instances in the data set are normal
  - **Supervised** : Labeled, imbalanced data set (normal/abnormal)
  - **Semi-Supervised** : Model representing normal behavior from a given normal training data set.
- e.g.) Bank fraud, Structual defect, System health monitoring, Intrusion detection, Fault detection, Ecosystem disturbances
- Source : [Wikipedia (https://goo.gl/YOdhxK)](https://goo.gl/YOdhxK)
- By [scikit-learn (https://goo.gl/csTPJr)](https://goo.gl/csTPJr) : **'*Training data contains outliers*'**

Novelty Detection

- **Mechanism by which an intelligent organism is able to identify an incoming sensory pattern as being hitherto unknown**
- The principle is long known in **neurophysiology(신경생리학)**
- 'Early neural modeling attempts were by Yehuda Salu(1988)'
- Source : [Wikipedia (https://goo.gl/6mntxw)](https://goo.gl/6mntxw)
- By [scikit-learn (https://goo.gl/csTPJr)](https://goo.gl/csTPJr) : **'*Training data is not polluted by outliers*'**

Evaluation Metrics for Anomaly & Novelty Detection

Metrics for Anomaly & Novelty Detection



ROC (Receiver Operating Characteristics)

- **False positive rate(FPR)** versus the **true positive rate(TPR)(=Recall)** for a number of different candidate threshold values between 0.0 and 1.0
- In other words, it plots the false alarm rate versus the hit rate

AUC (Area Under the Curve)

- Literally area under the ROC curve
- AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

PRC (Precision-Recall Curve)

- **Precision** versus the **Recall**
- To know how good a model is at predicting the positive class

Better metric for class-imbalanced data

- Precision captures false positive more sensitively than FPR, **thus PRC is more appropriate than ROC when it comes to class-imbalanced problem**

> e.g.)
> 1 million samples, 100 positive and others are all negative
> case1) 100 predicted positive, 90 true positive
> case2) 2000 predicted positive, 90 true positive
> case1) 0.9 TPR, 0.00001 FPR
> case2) 0.9 TPR, 0.00191 FPR
> FPR difference = 0.00190
>
> case1) 0.9 Recall, 0.9 Precision

About Dataset

- [NASA bearing dataset (acoustics.com/measurements/bearing-faults/bearing-4/)](acoustics.com/measurements/bearing-faults/bearing-4/)
- Made available by NASA
- Goals
  - to detect gear bearing degradation on an engine
  - to give a warning that allows for predictive measures to be taken in order to avoid a gear failure

Exploratory Data Analysis

In [1]:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
import tqdm
import copy
```

In [2]:

```
import ipywidgets as widgets
from ipywidgets import interact, interact_manual

sns.set(style='whitegrid', palette='muted')

%matplotlib inline
%load_ext autoreload
%autoreload 2
```

In [3]:

```
data_dir = os.path.abspath(os.path.join(os.getcwd(), 'data', '2nd_test'))
df = pd.DataFrame()

for filename in os.listdir(data_dir) :
    try :
        path = os.path.abspath(os.path.join(os.getcwd(), 'data', '2nd_test', fil
ename))
        dataset = pd.read_csv(path, sep='\t')

        dataset_mean_abs = np.array(dataset.abs().mean())
        dataset_mean_abs = pd.DataFrame(dataset_mean_abs.reshape(1,4))
        dataset_mean_abs.index = [filename]

        df = df.append(dataset_mean_abs)
    except Exception as e:
        print(e)

df.columns = ['Bearing 1', 'Bearing 2', 'Bearing 3', 'Bearing 4']
```

```
Error tokenizing data. C error: Expected 2 fields in line 4, saw 5
```

In [4]:

```
df.index = pd.to_datetime(df.index, format='%Y.%m.%d.%H.%M.%S')
df = df.sort_index()
```

In [5]:

```
@interact
def show(x=df.shape[0]) :
    return df.iloc[x:x+10]
```

In [6]:

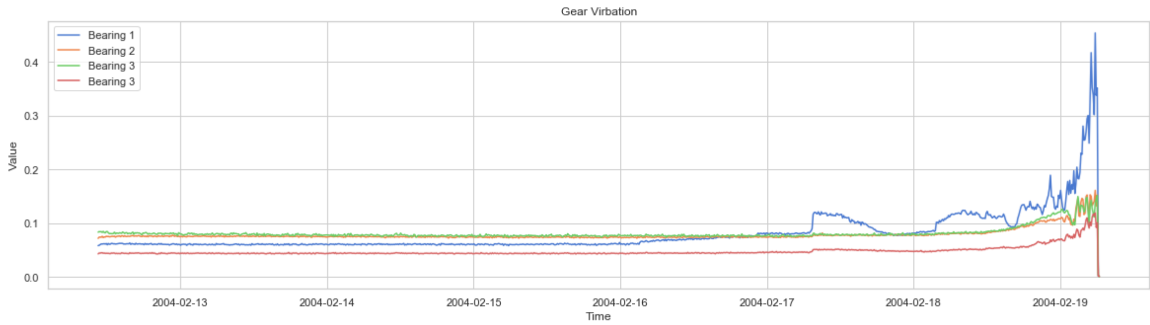
```
@interact
def plot(x=df.shape[0]) :
    plt.figure(figsize=(20,5))
    plt.plot(df['Bearing 1'].iloc[x:x+100])
    plt.plot(df['Bearing 2'].iloc[x:x+100])
    plt.plot(df['Bearing 3'].iloc[x:x+100])
    plt.plot(df['Bearing 4'].iloc[x:x+100])
    plt.legend()
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.title('Gear Virbation')
    plt.show()
```

In [7]:

```
def show_all_period_plot(df) :
    plt.figure(figsize=(20, 5))
    plt.plot(df['Bearing 1'], label='Bearing 1')
    plt.plot(df['Bearing 2'], label='Bearing 2')
    plt.plot(df['Bearing 3'], label='Bearing 3')
    plt.plot(df['Bearing 4'], label='Bearing 3')
    plt.legend()
    plt.xlabel('Time')
    plt.ylabel('Value')
    plt.title('Gear Virbation')
    plt.show()
```

In [8]:

```
show_all_period_plot(df)
```



In [9]:

```
x_train = df[:'2004-02-16 22:02:39']
x_valid = df['2004-02-16 22:02:39':]
```

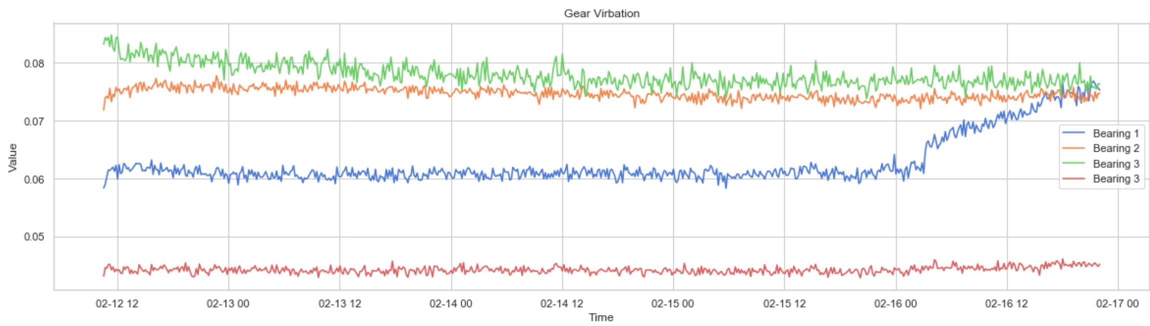
In [10]:

```
print(x_train.shape)
print(x_valid.shape)
```

```
(646, 4)
(339, 4)
```

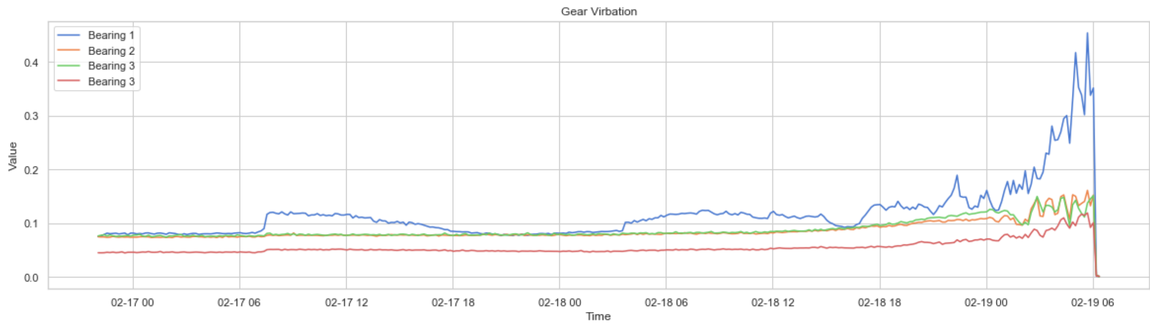
In [11]:

```
show_all_period_plot(x_train)
```



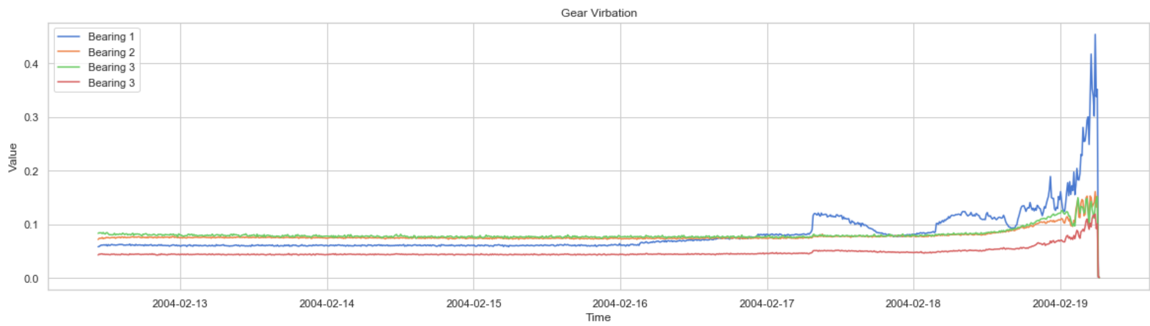
In [12]:

```
show_all_period_plot(x_valid)
```



In [13]:

```
show_all_period_plot(pd.concat([x_train, x_valid]))
```

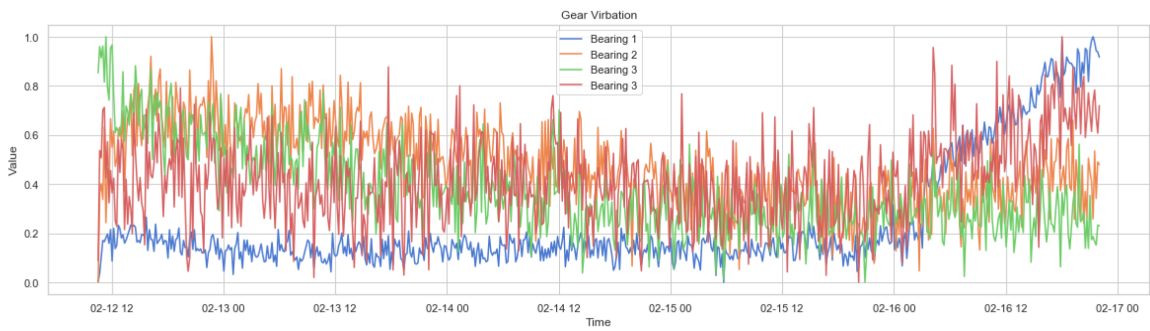


In [14]:

```
minmaxscaler = MinMaxScaler()

x_train = pd.DataFrame(minmaxscaler.fit_transform(x_train), columns=x_train.colu
mns, index=x_train.index)
x_valid = pd.DataFrame(minmaxscaler.transform(x_valid), columns=x_valid.columns,
index=x_valid.index)
```

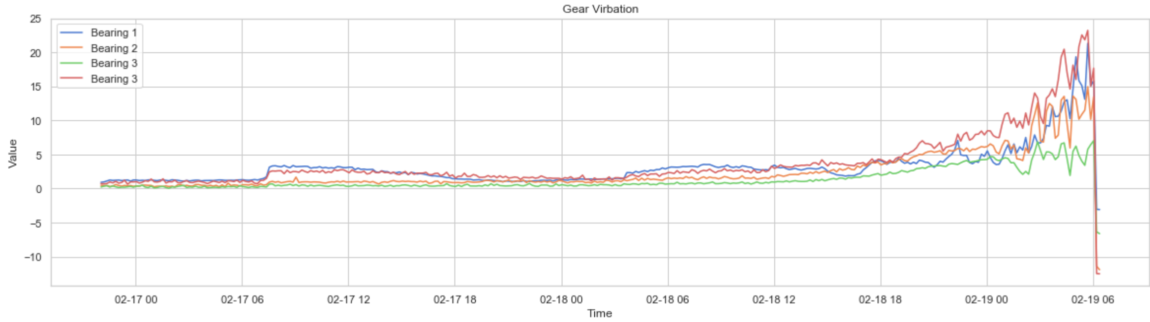
In [15]:

```
show_all_period_plot(x_train)
```



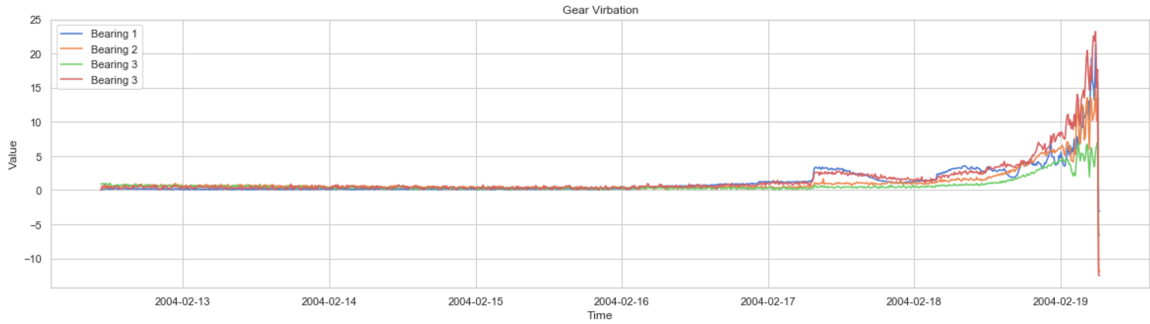
In [16]:

```
show_all_period_plot(x_valid)
```



In [17]:

```
show_all_period_plot(pd.concat([x_train, x_valid]))
```



Novelty Detection using PCA Reconstruction

In [18]:

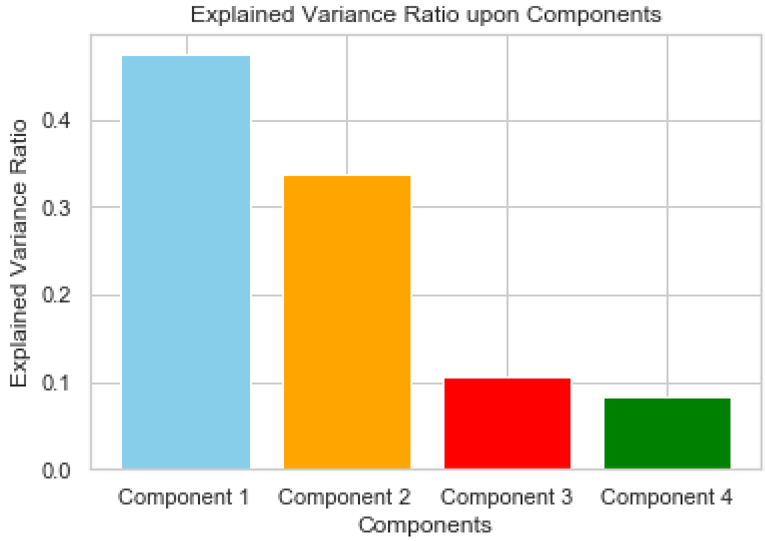
```
pca = PCA(n_components=4)
pca.fit(x_train)
```

Out[18]:

```
PCA(copy=True, iterated_power='auto', n_components=4, random_state=N
one,
  svd_solver='auto', tol=0.0, whiten=False)
```

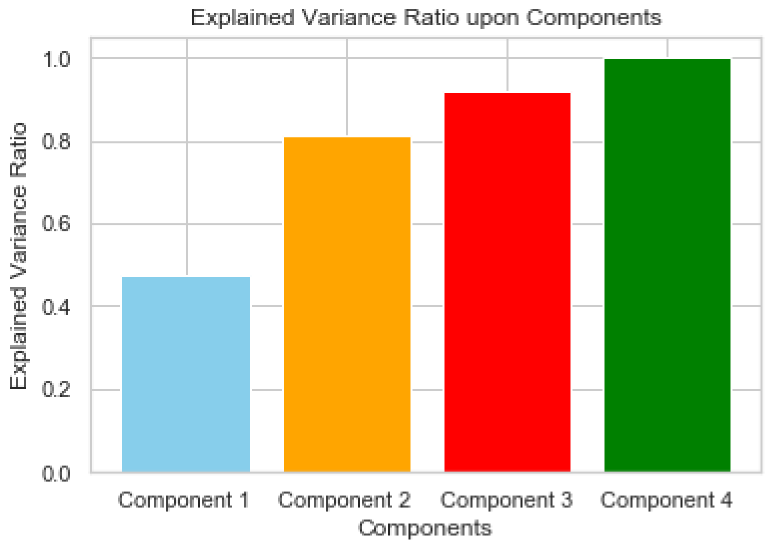
In [19]:

```
plt.bar(['Component 1', 'Component 2', 'Component 3', 'Component 4'], pca.explai
ned_variance_ratio_, color=['skyblue', 'orange', 'red', 'green'])
plt.xlabel('Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio upon Components')
plt.show()
```



In [20]:

```
plt.bar(['Component 1', 'Component 2', 'Component 3', 'Component 4'], np.cumsum(
pca.explained_variance_ratio_), color=['skyblue', 'orange', 'red', 'green'])
plt.xlabel('Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio upon Components')
plt.show()
print(np.cumsum(pca.explained_variance_ratio_))
```




```
[0.4746206  0.81127943 0.91640243 1.        ]
```

- Component2까지 사용했을때 Variance의 73%,
- Component3까지 사용했을때 Variance의 90%를 설명하는 것을 볼 수 있다

Number of Components = 2

In [21]:

```
pca = PCA(n_components=2, svd_solver='full')

x_train_pca = pd.DataFrame(pca.fit_transform(x_train), index=x_train.index)
x_valid_pca = pd.DataFrame(pca.transform(x_valid), index=x_valid.index)
```

In [22]:

```
print(x_train_pca.shape)
print(x_valid_pca.shape)
```

```
(646, 2)
(339, 2)
```

In [23]:

```
display(x_train_pca.head())
display(x_valid_pca.head())
```

	0	1
2004-02-12 10:32:39	-0.379091	-0.165629
2004-02-12 10:42:39	-0.257896	0.322483
2004-02-12 10:52:39	-0.210759	0.313994
2004-02-12 11:02:39	-0.070890	0.397639
2004-02-12 11:12:39	-0.088929	0.489085

	0	1
2004-02-16 22:02:39	0.733423	0.142553
2004-02-16 22:12:39	0.743653	0.185636
2004-02-16 22:22:39	0.849600	0.289164
2004-02-16 22:32:39	1.156566	0.190001
2004-02-16 22:42:39	0.984879	0.272958

In [24]:

```
x_train_pca_recon = pd.DataFrame(pca.inverse_transform(x_train_pca), index=x_tra
in.index, columns=df.columns)
x_valid_pca_recon = pd.DataFrame(pca.inverse_transform(x_valid_pca), index=x_val
id.index, columns=df.columns)
```

In [25]:

```
print(x_train_pca_recon.shape)
print(x_valid_pca_recon.shape)
```

```
(646, 4)
(339, 4)
```

In [26]:

```
x_train_mse = (x_train_pca_recon - x_train)
x_valid_mse = (x_valid_pca_recon - x_valid)
```

In [27]:

```
x_train_mse['Reconstruction Error'] = x_train_mse.mean(axis=1).apply(lambda x :
x*x)
x_valid_mse['Reconstruction Error'] = x_valid_mse.mean(axis=1).apply(lambda x :
x*x)
```

In [28]:

```
display(x_train.head(),)
display(x_train_pca_recon.head())
display(x_train_mse.head())
```

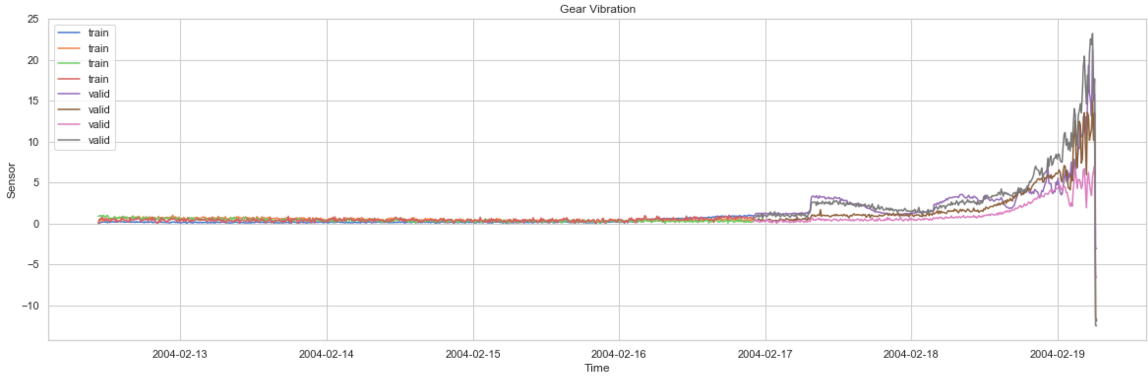
	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:32:39	0.002152	0.000000	0.851843	0.088117
2004-02-12 10:42:39	0.037843	0.364172	0.959840	0.536515
2004-02-12 10:52:39	0.104727	0.401176	0.913750	0.506727
2004-02-12 11:02:39	0.170414	0.337099	0.961833	0.700840
2004-02-12 11:12:39	0.165312	0.632695	0.815186	0.712082

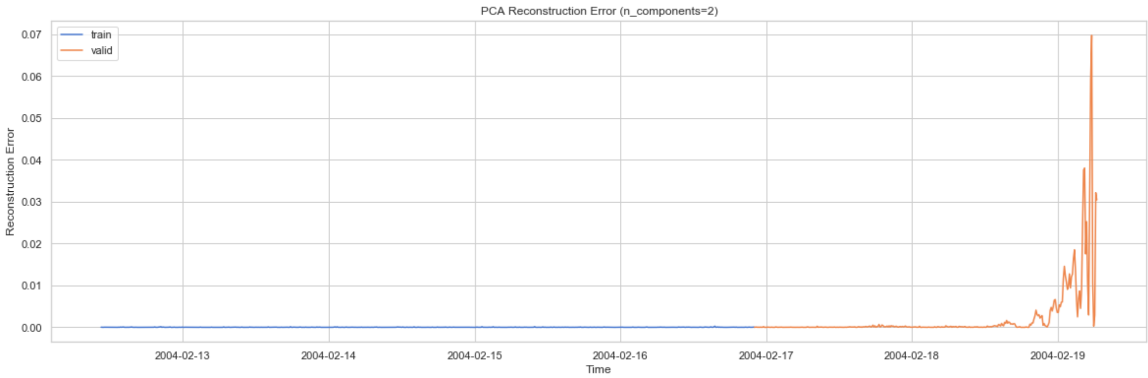
	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:32:39	-0.100647	0.456851	0.382074	0.180047
2004-02-12 10:42:39	0.078229	0.732311	0.661472	0.438898
2004-02-12 10:52:39	0.115342	0.717134	0.642026	0.456452
2004-02-12 11:02:39	0.243277	0.739220	0.654628	0.554155
2004-02-12 11:12:39	0.243509	0.799419	0.719040	0.584400

	Bearing 1	Bearing 2	Bearing 3	Bearing 4	Reconstruction Error
2004-02-12 10:32:39	-0.102800	0.456851	-0.469768	0.091930	0.000035
2004-02-12 10:42:39	0.040386	0.368138	-0.298368	-0.097617	0.000010
2004-02-12 10:52:39	0.010614	0.315958	-0.271724	-0.050274	0.000001
2004-02-12 11:02:39	0.072863	0.402122	-0.307205	-0.146686	0.000028
2004-02-12 11:12:39	0.078197	0.166724	-0.096147	-0.127681	0.000028

In [29]:

```
plt.figure(figsize=(20,6))
plt.plot(x_train, label='train')
plt.plot(x_valid, label='valid')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Sensor')
plt.title('Gear Vibration')
plt.show()

plt.figure(figsize=(20, 6))
plt.plot(x_train_mse['Reconstruction Error'], label='train')
plt.plot(x_valid_mse['Reconstruction Error'], label='valid')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Reconstruction Error')
plt.title('PCA Reconstruction Error (n_components=2)')
plt.show()
```





Number of Components = 3

In [30]:

```
pca = PCA(n_components=3, svd_solver='full')

x_train_pca = pd.DataFrame(pca.fit_transform(x_train), index=x_train.index)
x_valid_pca = pd.DataFrame(pca.transform(x_valid), index=x_valid.index)
```

In [31]:

```
print(x_train_pca.shape)
print(x_valid_pca.shape)
```

```
(646, 3)
(339, 3)
```

In [32]:

```
display(x_train_pca.head())
display(x_valid_pca.head())
```

	0	1	2
2004-02-12 10:32:39	-0.379091	-0.165629	0.222976
2004-02-12 10:42:39	-0.257896	0.322483	-0.044927
2004-02-12 10:52:39	-0.210759	0.313994	0.005273
2004-02-12 11:02:39	-0.070890	0.397639	-0.101475
2004-02-12 11:12:39	-0.088929	0.489085	-0.129541

	0	1	2
2004-02-16 22:02:39	0.733423	0.142553	0.100286
2004-02-16 22:12:39	0.743653	0.185636	0.176447
2004-02-16 22:22:39	0.849600	0.289164	0.248469
2004-02-16 22:32:39	1.156566	0.190001	0.071281
2004-02-16 22:42:39	0.984879	0.272958	0.197234

In [33]:

```
x_train_pca_recon = pd.DataFrame(pca.inverse_transform(x_train_pca), index=x_tra
in.index, columns=df.columns)
x_valid_pca_recon = pd.DataFrame(pca.inverse_transform(x_valid_pca), index=x_val
id.index, columns=df.columns)
```

In [34]:

```
print(x_train_pca_recon.shape)
print(x_valid_pca_recon.shape)
```

```
(646, 4)
(339, 4)
```

In [35]:

```
x_train_mse = (x_train_pca_recon - x_train)
x_valid_mse = (x_valid_pca_recon - x_valid)
```

In [36]:

```
x_train_mse['Reconstruction Error'] = x_train_mse.mean(axis=1).apply(lambda x :
x*x)
x_valid_mse['Reconstruction Error'] = x_valid_mse.mean(axis=1).apply(lambda x :
x*x)
```

In [37]:

```
display(x_train.head(),)
display(x_train_pca_recon.head())
display(x_train_mse.head())
```

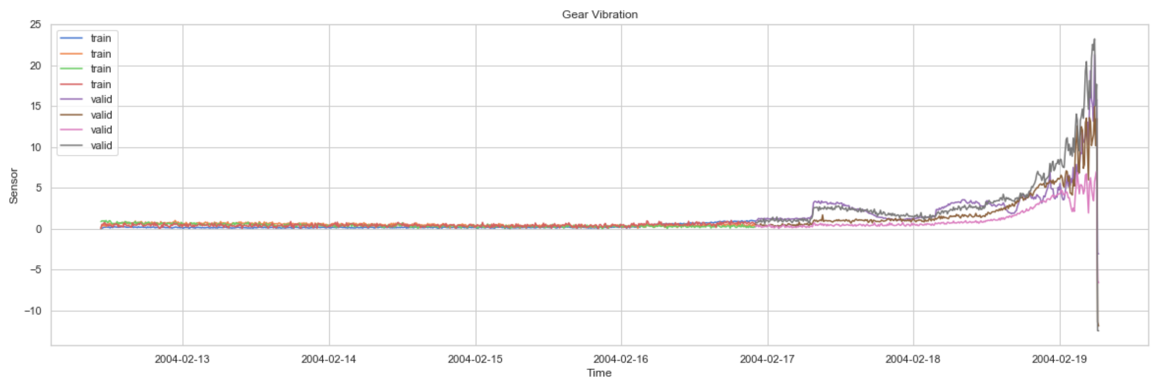
	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:32:39	0.002152	0.000000	0.851843	0.088117
2004-02-12 10:42:39	0.037843	0.364172	0.959840	0.536515
2004-02-12 10:52:39	0.104727	0.401176	0.913750	0.506727
2004-02-12 11:02:39	0.170414	0.337099	0.961833	0.700840
2004-02-12 11:12:39	0.165312	0.632695	0.815186	0.712082

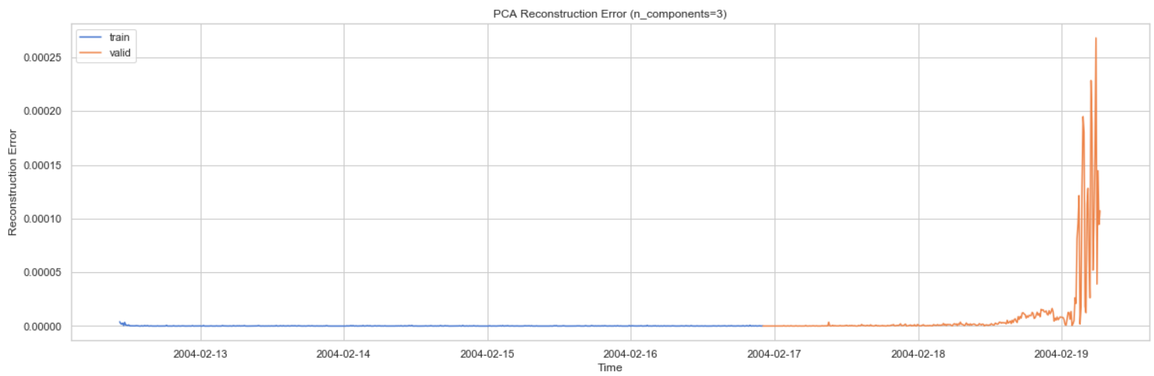
	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:32:39	0.022495	0.475850	0.445445	0.006331
2004-02-12 10:42:39	0.053417	0.728483	0.648703	0.473899
2004-02-12 10:52:39	0.118254	0.717584	0.643525	0.452344
2004-02-12 11:02:39	0.187236	0.730573	0.625788	0.633212
2004-02-12 11:12:39	0.171968	0.788381	0.682224	0.685323

	Bearing 1	Bearing 2	Bearing 3	Bearing 4	Reconstruction Error
2004-02-12 10:32:39	0.020343	0.475850	-0.406397	-0.081787	4.009175e-06
2004-02-12 10:42:39	0.015574	0.364310	-0.311137	-0.062616	2.349941e-06
2004-02-12 10:52:39	0.013526	0.316407	-0.270226	-0.054383	1.772586e-06
2004-02-12 11:02:39	0.016821	0.393475	-0.336045	-0.067628	2.741246e-06
2004-02-12 11:12:39	0.006656	0.155686	-0.132963	-0.026759	4.291557e-07

In [38]:

```
plt.figure(figsize=(20,6))
plt.plot(x_train, label='train')
plt.plot(x_valid, label='valid')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Sensor')
plt.title('Gear Vibration')
plt.show()

plt.figure(figsize=(20, 6))
plt.plot(x_train_mse['Reconstruction Error'], label='train')
plt.plot(x_valid_mse['Reconstruction Error'], label='valid')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Reconstruction Error')
plt.title('PCA Reconstruction Error (n_components=3)')
plt.show()
```





Conclusion

- The features of normal data were trained by usig normal data only. Anomaly could be dected by using reconstruction error inferenced.
- PCA Reconstruction compressed 4 features as single value.
- PCA Reconstruction error plot showed similar trend with original sensor plot, which means PCA reconstruction error well summarized given data.
- If the labels of defect were given, the model could be evaluated by AUROC which distinguishes normal/abrnomal by using reconstruction error.
- As labels didn't exist, threshold for normal/abnormal couldn't set.
- It's not worthy to set threshold without knowing which samples are defect, hence it's not done.
- **Data will be given in the future will have labels, thus model can be evaluted by AUROC, AUPRC**

Novelty Detection using VAE Reconsturction Error

In [40]:

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
batch_size= 16
input_dim = 4
epochs = 3000
lr = 1e-3
```

In [41]:

```
x_train = df[:'2004-02-16 22:02:39']
x_valid = df['2004-02-16 22:02:39':'2004-02-17 21:12:39']
x_test = df['2004-02-17 21:12:39':]
```

In [42]:

```
minmaxscaler = MinMaxScaler()

x_train = pd.DataFrame(minmaxscaler.fit_transform(x_train), columns=x_train.colu
mns, index=x_train.index)
x_valid = pd.DataFrame(minmaxscaler.transform(x_valid), columns=x_valid.columns,
index=x_valid.index)
x_test = pd.DataFrame(minmaxscaler.transform(x_test), columns=x_test.columns, in
dex=x_test.index)
```

In [43]:

```
x_train_torch = torch.from_numpy(x_train.values).float()
x_valid_torch = torch.from_numpy(x_valid.values).float()
x_test_torch = torch.from_numpy(x_test.values).float()
```

In [44]:

```
train_loader = DataLoader(x_train_torch, batch_size=batch_size, shuffle=True)
valid_loader = DataLoader(x_valid_torch, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(x_test_torch, batch_size=batch_size, shuffle=True)
```

Bottleneck Dimension = 2

In [45]:

```
z_dim = 2
```

In [46]:

```
class VAE(nn.Module) :
    def __init__(self) :
        super(VAE,self).__init__()
        self.fc1 = nn.Linear(4, 10)
        self.fc21 = nn.Linear(10, 2)
        self.fc22 = nn.Linear(10, 2)

        self.fc3 = nn.Linear(2, 10)
        self.fc4 = nn.Linear(10, 4)
        self.relu = nn.ReLU()

    def encode(self, x) :
        h1 = self.relu(self.fc1(x))
        return self.fc21(h1), self.fc22(h1)

    def reparameterize(self, mu, log_var) :
        std = torch.exp(log_var/2)
        eps = torch.rand_like(std)
        return mu+eps*std

    def decode(self, z) :
        h2 = self.relu(self.fc3(z))
        return self.fc4(h2)

    def forward(self, x) :
        mu, log_var = self.encode(x)
        z = self.reparameterize(mu, log_var)
        x_recon = self.decode(z)
        return x_recon, mu, log_var
```

In [47]:

```
model = VAE().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

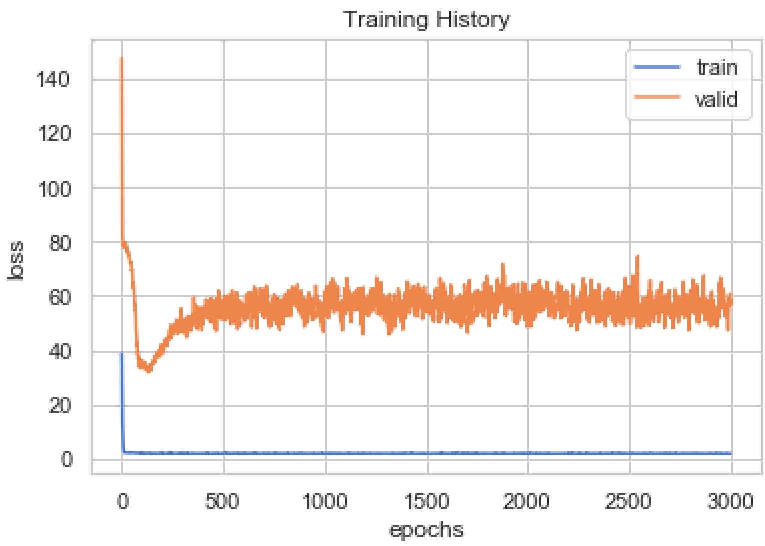
In [48]:

```
train_loss = []
valid_loss = []
best_model = copy.deepcopy(model)
lowest_loss = 10000

outer = tqdm.tqdm(total=epochs, desc='Epoch', position=0)

for epoch in range(epochs) :
    # train
    loss_per_epoch = 0
    model.train()

    for x in train_loader :
        x_recon, mu, log_var = model(x)

        recon_loss = F.mse_loss(x_recon, x, reduction='sum')
        kld = -0.5 * torch.sum(1+log_var-mu.pow(2)-log_var.exp())
        loss = recon_loss + kld
        loss_per_epoch += loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    train_loss.append(loss_per_epoch / (len(train_loader.dataset)/batch_size) )

    # validation
    loss_per_epoch = 0
    model.eval()
    with torch.no_grad() :
        for x in valid_loader :
            x_recon , mu, log_var = model(x)

            recon_loss = F.mse_loss(x_recon, x, reduction='sum')
            kld = -0.5 * torch.sum(1+log_var-mu.pow(2)-log_var.exp())
            loss = recon_loss + kld
            loss_per_epoch += recon_loss.item() + kld
    current_valid_loss = loss_per_epoch / (len(valid_loader.dataset)/batch_size)
    valid_loss.append(current_valid_loss)

    # save best model
    if epoch == 0 :
        lowest_loss = current_valid_loss
    else :
        if current_valid_loss <= lowest_loss :
            lowest_loss = current_valid_loss
            best_model = copy.deepcopy(model)

    outer.update(1)
```

```
Epoch: 100%|██████████| 2999/3000 [04:51<00:00, 15.16it/s]
```

In [49]:

```
plt.plot(train_loss, label='train')
plt.plot(valid_loss, label='valid')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.title('Training History')
plt.show()
```



In [50]:

```
train_recon = best_model(x_train_torch)[0]
valid_recon = best_model(x_valid_torch)[0]
test_recon = best_model(x_test_torch)[0]
```

In [51]:

```
x_train_vae_recon = pd.DataFrame(train_recon.detach().numpy(), columns=x_train.c
olumns, index=x_train.index)
x_valid_vae_recon = pd.DataFrame(valid_recon.detach().numpy(), columns=x_valid.c
olumns, index=x_valid.index)
x_test_vae_recon = pd.DataFrame(test_recon.detach(). numpy(), columns=x_test.col
umns, index=x_test.index)
```

	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:32:39	0.205759	0.516963	0.421281	0.420528
2004-02-12 10:42:39	0.182593	0.517504	0.425911	0.404980
2004-02-12 10:52:39	0.219581	0.433184	0.337692	0.397972
2004-02-12 11:02:39	0.142171	0.475738	0.388426	0.356774
2004-02-12 11:12:39	0.138406	0.561779	0.480876	0.396666

In [52]:

```
x_train_mse = (x_train_vae_recon - x_train)
x_valid_mse = (x_valid_vae_recon - x_valid)
x_test_mse = (x_test_vae_recon - x_test)
```

In [53]:

```
x_train_mse['Reconstruction Error'] = x_train_mse.mean(axis=1).apply(lambda x :
x*x)
x_valid_mse['Reconstruction Error'] = x_valid_mse.mean(axis=1).apply(lambda x :
x*x)
x_test_mse['Reconstruction Error'] = x_test_mse.mean(axis=1).apply(lambda x : x*
x)
```

In [54]:

```
display(x_train.head(),)
display(x_train_vae_recon.head())
display(x_train_mse.head())
```

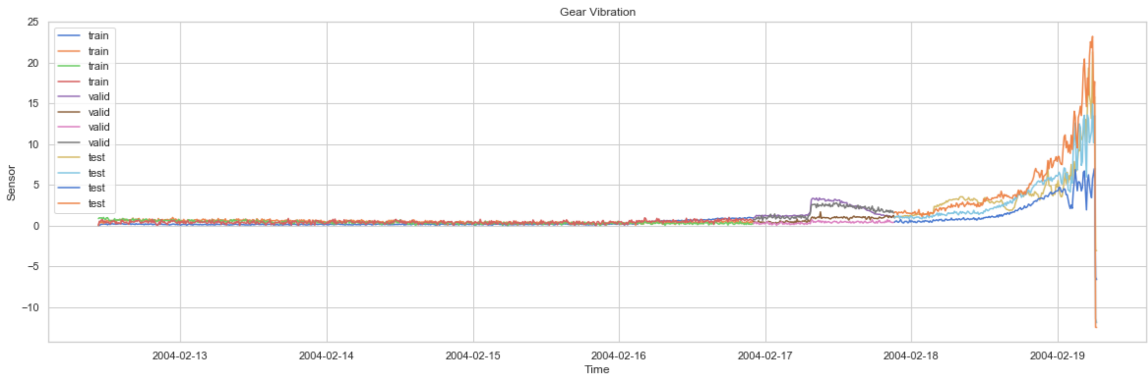
	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:32:39	0.002152	0.000000	0.851843	0.088117
2004-02-12 10:42:39	0.037843	0.364172	0.959840	0.536515
2004-02-12 10:52:39	0.104727	0.401176	0.913750	0.506727
2004-02-12 11:02:39	0.170414	0.337099	0.961833	0.700840
2004-02-12 11:12:39	0.165312	0.632695	0.815186	0.712082

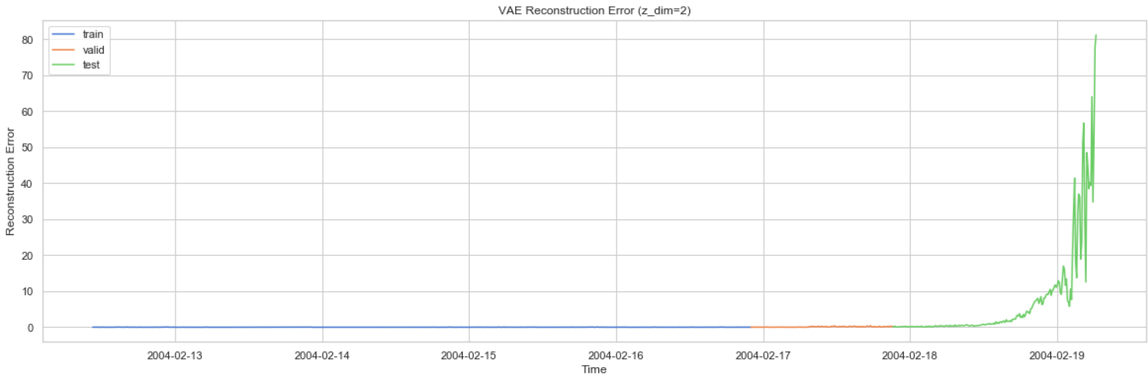
	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:32:39	0.205759	0.516963	0.421281	0.420528
2004-02-12 10:42:39	0.182593	0.517504	0.425911	0.404980
2004-02-12 10:52:39	0.219581	0.433184	0.337692	0.397972
2004-02-12 11:02:39	0.142171	0.475738	0.388426	0.356774
2004-02-12 11:12:39	0.138406	0.561779	0.480876	0.396666

	Bearing 1	Bearing 2	Bearing 3	Bearing 4	Reconstruction Error
2004-02-12 10:32:39	0.203606	0.516963	-0.430561	0.332410	0.024213
2004-02-12 10:42:39	0.144750	0.153331	-0.533929	-0.131535	0.008436
2004-02-12 10:52:39	0.114854	0.032008	-0.576058	-0.108755	0.018087
2004-02-12 11:02:39	-0.028243	0.138639	-0.573407	-0.344066	0.040711
2004-02-12 11:12:39	-0.026907	-0.070915	-0.334311	-0.315415	0.034927

In [55]:

```
plt.figure(figsize=(20,6))
plt.plot(x_train, label='train')
plt.plot(x_valid, label='valid')
plt.plot(x_test, label='test')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Sensor')
plt.title('Gear Vibration')
plt.show()

plt.figure(figsize=(20, 6))
plt.plot(x_train_mse['Reconstruction Error'], label='train')
plt.plot(x_valid_mse['Reconstruction Error'], label='valid')
plt.plot(x_test_mse['Reconstruction Error'], label='test')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Reconstruction Error')
plt.title('VAE Reconstruction Error (z_dim=2)')
plt.show()
```





Bottleneck Dimension = 3

In [56]:

```
z_dim = 3
```

In [57]:

```
class VAE(nn.Module) :
    def __init__(self) :
        super(VAE,self).__init__()
        self.fc1 = nn.Linear(4, 10)
        self.fc21 = nn.Linear(10, 3)
        self.fc22 = nn.Linear(10, 3)

        self.fc3 = nn.Linear(3, 10)
        self.fc4 = nn.Linear(10, 4)
        self.relu = nn.ReLU()

    def encode(self, x) :
        h1 = self.relu(self.fc1(x))
        return self.fc21(h1), self.fc22(h1)

    def reparameterize(self, mu, log_var) :
        std = torch.exp(log_var/2)
        eps = torch.rand_like(std)
        return mu+eps*std

    def decode(self, z) :
        h2 = self.relu(self.fc3(z))
        return self.fc4(h2)

    def forward(self, x) :
        mu, log_var = self.encode(x)
        z = self.reparameterize(mu, log_var)
        x_recon = self.decode(z)
        return x_recon, mu, log_var
```

In [58]:

```
model = VAE().to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

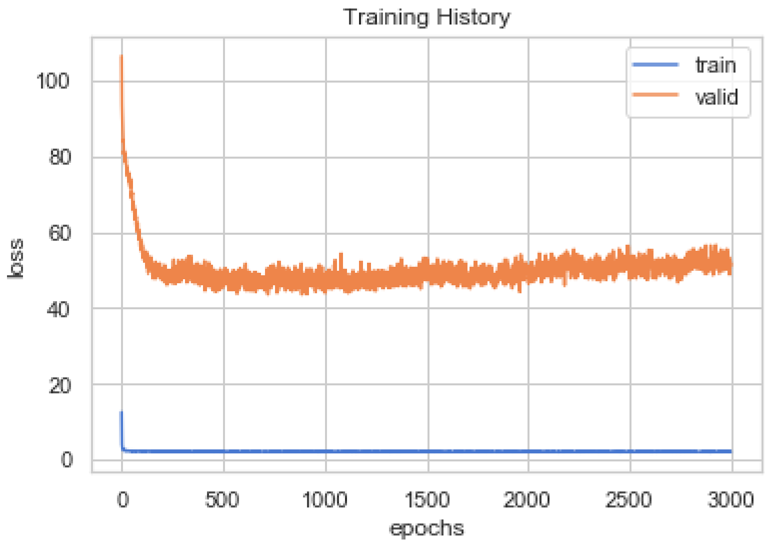
In [59]:

```
train_loss = []
valid_loss = []
best_model = copy.deepcopy(model)
lowest_loss = 10000

outer = tqdm.tqdm(total=epochs, desc='Epoch', position=0)

for epoch in range(epochs) :
    # train
    loss_per_epoch = 0
    model.train()

    for x in train_loader :
        x_recon, mu, log_var = model(x)

        recon_loss = F.mse_loss(x_recon, x, reduction='sum')
        kld = -0.5 * torch.sum(1+log_var-mu.pow(2)-log_var.exp())
        loss = recon_loss + kld
        loss_per_epoch += loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    train_loss.append(loss_per_epoch / (len(train_loader.dataset)/batch_size) )

    # validation
    loss_per_epoch = 0
    model.eval()
    with torch.no_grad() :
        for x in valid_loader :
            x_recon , mu, log_var = model(x)

            recon_loss = F.mse_loss(x_recon, x, reduction='sum')
            kld = -0.5 * torch.sum(1+log_var-mu.pow(2)-log_var.exp())
            loss = recon_loss + kld
            loss_per_epoch += recon_loss.item() + kld
    current_valid_loss = loss_per_epoch / (len(valid_loader.dataset)/batch_size)
    valid_loss.append(current_valid_loss)

    # save best model
    if epoch == 0 :
        lowest_loss = current_valid_loss
    else :
        if current_valid_loss <= lowest_loss :
            lowest_loss = current_valid_loss
            best_model = copy.deepcopy(model)

    outer.update(1)
```

```
Epoch: 100%|██████████| 2999/3000 [04:29<00:00, 12.68it/s]
```

In [60]:

```
plt.plot(train_loss, label='train')
plt.plot(valid_loss, label='valid')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.title('Training History')
plt.show()
```



In [61]:

```
train_recon = best_model(x_train_torch)[0]
valid_recon = best_model(x_valid_torch)[0]
test_recon = best_model(x_test_torch)[0]
```

In [62]:

```
x_train_vae_recon = pd.DataFrame(train_recon.detach().numpy(), columns=x_train.c
olumns, index=x_train.index)
x_valid_vae_recon = pd.DataFrame(valid_recon.detach().numpy(), columns=x_valid.c
olumns, index=x_valid.index)
x_test_vae_recon = pd.DataFrame(test_recon.detach(). numpy(), columns=x_test.col
umns, index=x_test.index)
```

	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:32:39	0.274581	0.454213	0.344379	0.446383
2004-02-12 10:42:39	0.291919	0.478838	0.369266	0.464424
2004-02-12 10:52:39	0.136212	0.494747	0.406424	0.374942
2004-02-12 11:02:39	0.137311	0.590052	0.526276	0.423193
2004-02-12 11:12:39	0.235123	0.506643	0.400674	0.449933

In [63]:

```
x_train_mse = (x_train_vae_recon - x_train)
x_valid_mse = (x_valid_vae_recon - x_valid)
x_test_mse = (x_test_vae_recon - x_test)
```

In [64]:

```
x_train_mse['Reconstruction Error'] = x_train_mse.mean(axis=1).apply(lambda x :
x*x)
x_valid_mse['Reconstruction Error'] = x_valid_mse.mean(axis=1).apply(lambda x :
x*x)
x_test_mse['Reconstruction Error'] = x_test_mse.mean(axis=1).apply(lambda x : x*
x)
```

In [65]:

```
display(x_train.head(),)
display(x_train_vae_recon.head())
display(x_train_mse.head())
```

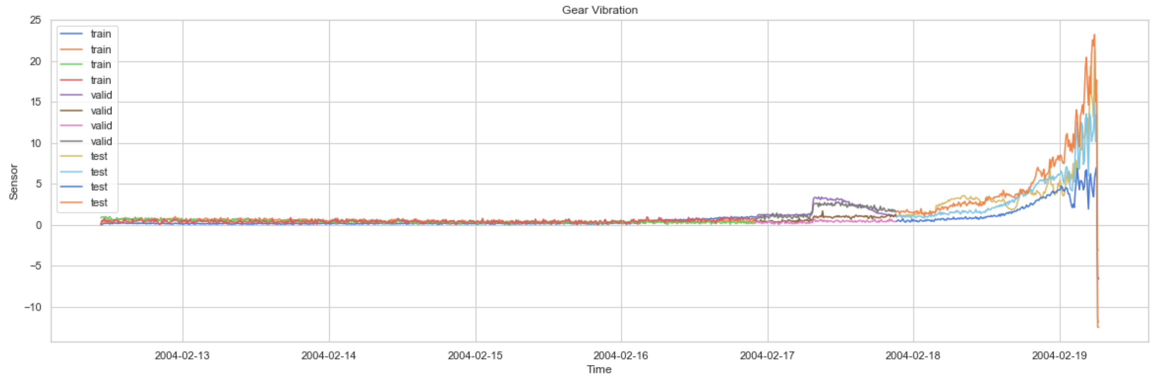
	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:32:39	0.002152	0.000000	0.851843	0.088117
2004-02-12 10:42:39	0.037843	0.364172	0.959840	0.536515
2004-02-12 10:52:39	0.104727	0.401176	0.913750	0.506727
2004-02-12 11:02:39	0.170414	0.337099	0.961833	0.700840
2004-02-12 11:12:39	0.165312	0.632695	0.815186	0.712082

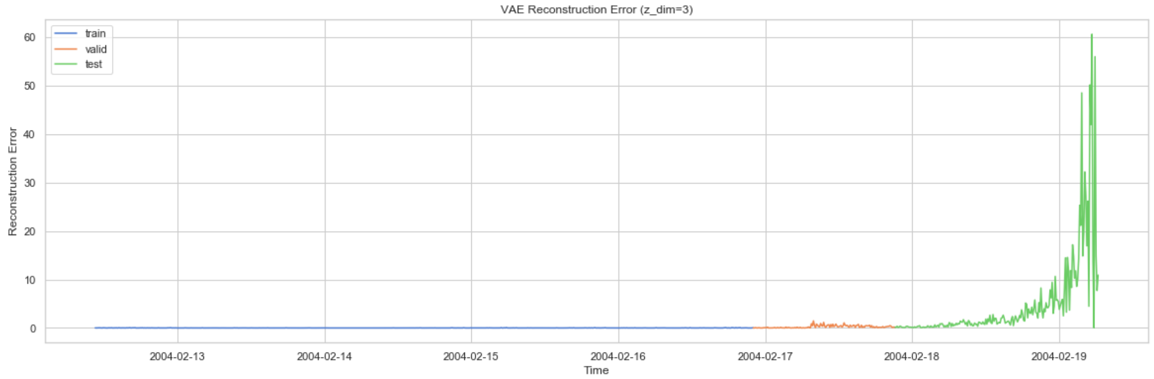
	Bearing 1	Bearing 2	Bearing 3	Bearing 4
2004-02-12 10:32:39	0.274581	0.454213	0.344379	0.446383
2004-02-12 10:42:39	0.291919	0.478838	0.369266	0.464424
2004-02-12 10:52:39	0.136212	0.494747	0.406424	0.374942
2004-02-12 11:02:39	0.137311	0.590052	0.526276	0.423193
2004-02-12 11:12:39	0.235123	0.506643	0.400674	0.449933

	Bearing 1	Bearing 2	Bearing 3	Bearing 4	Reconstruction Error
2004-02-12 10:32:39	0.272428	0.454213	-0.507464	0.358265	0.020840
2004-02-12 10:42:39	0.254076	0.114666	-0.590574	-0.072091	0.005399
2004-02-12 10:52:39	0.031484	0.093571	-0.507326	-0.131785	0.016516
2004-02-12 11:02:39	-0.033104	0.252954	-0.435556	-0.277647	0.015212
2004-02-12 11:12:39	0.069810	-0.126052	-0.414512	-0.262148	0.033572

In [66]:

```
plt.figure(figsize=(20,6))
plt.plot(x_train, label='train')
plt.plot(x_valid, label='valid')
plt.plot(x_test, label='test')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Sensor')
plt.title('Gear Vibration')
plt.show()

plt.figure(figsize=(20, 6))
plt.plot(x_train_mse['Reconstruction Error'], label='train')
plt.plot(x_valid_mse['Reconstruction Error'], label='valid')
plt.plot(x_test_mse['Reconstruction Error'], label='test')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Reconstruction Error')
plt.title('VAE Reconstruction Error (z_dim=3)')
plt.show()
```





Conclusion

- The features of normal data were trained by usig normal data only. Anomaly could be dected by using reconstruction error inferenced.
- VAE Reconstruction compressed 4 features as single value.
- In order to prevent overfitting, dataset was spllited as train, valid, test dataset. Best model was saved since loss of validation set, which has similiar distribution with training set, started increasing.
- VAE Reconstruction error plot showed similar trend with original sensor plot, which means VAE reconstruction error well summarized given data.
- If the labels of defect were given, the model could be evaluated by AUROC which distinguishes normal/abrnomal by using reconstruction error.
- As labels didn't exist, threshold for normal/abnormal couldn't set.
- It's not worthy to set threshold without knowing which samples are defect, hence it's not done.
- **Data will be given in the future will have labels, thus model can be evaluted by AUROC, AUPRC**