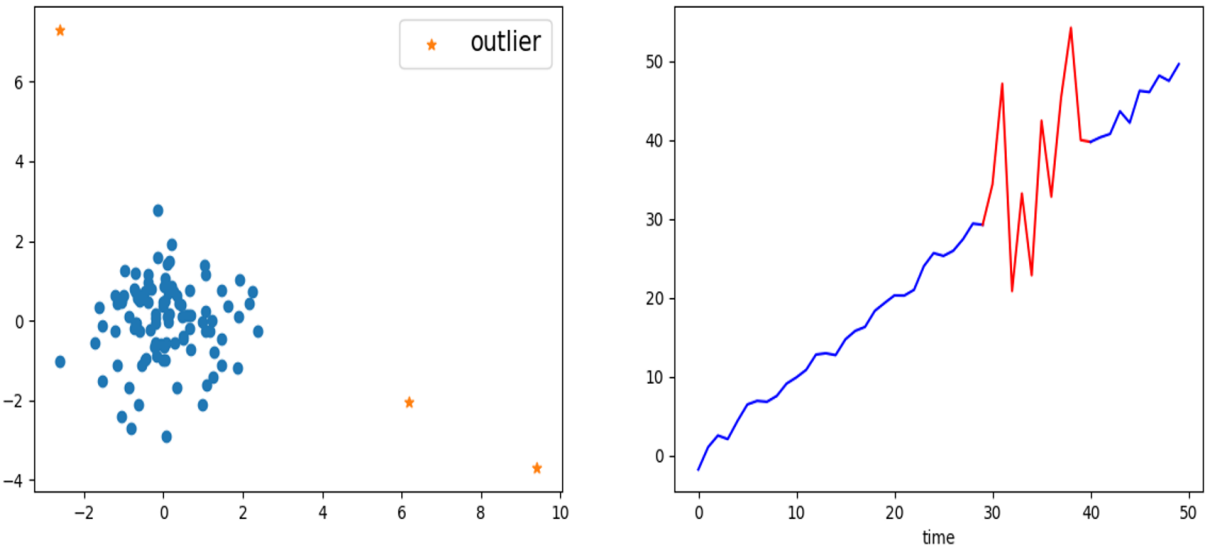
LSTM for Anomaly Detection in time series

Anomaly detection of time series using LSTM

Anomaly detection has applications to many fields, such as system health monitoring, fraud detection, and intrusion detection. Among them, there are two types of anomalies.



One type of anomaly is known as 'outlier', which is a value located outside of the normal class, as shown on the left side of the figure above. The other type of anomaly is an anomalous behavior, which is a periodic collapsing phenomenon in time series, as shown on the right side of the figure above. Even when an anomalous behavior gets a normal value, it is an anomaly in terms of a periodicity.
LSTM is the neural network that can be applied to the time-series analysis.
In this tutorial, we will explain the latter type of anomaly detection using LSTM.

Algorithm

In this tutorial, we use the anomaly detection algorithm proposed in
"Long short term memory networks for anomaly detection in time series.",Malhotra, Pankaj, et al, 2015.
The algorithm consists of three steps.

STEP1

Train LSTM to predict the next l values $\{x_{t+1}, \ldots, x_{t+l}\}$ from the previous d data $\{x_{t-d+1}, \ldots, x_t\}$. The figure below represents the case of $d = 2, l = 1$.
When using the time series $\{x_1, \ldots, x_T\}$, the input of LSTM is a sequence of M dimensional vectors $\{x_{t-d+1}, \ldots, x_t\}$ and the output is l vectors of M dimensional vector $\{x_{t+1}, \ldots, x_{t+l}\}$ which is predicted at once.

STEP2

Compute error vectors :

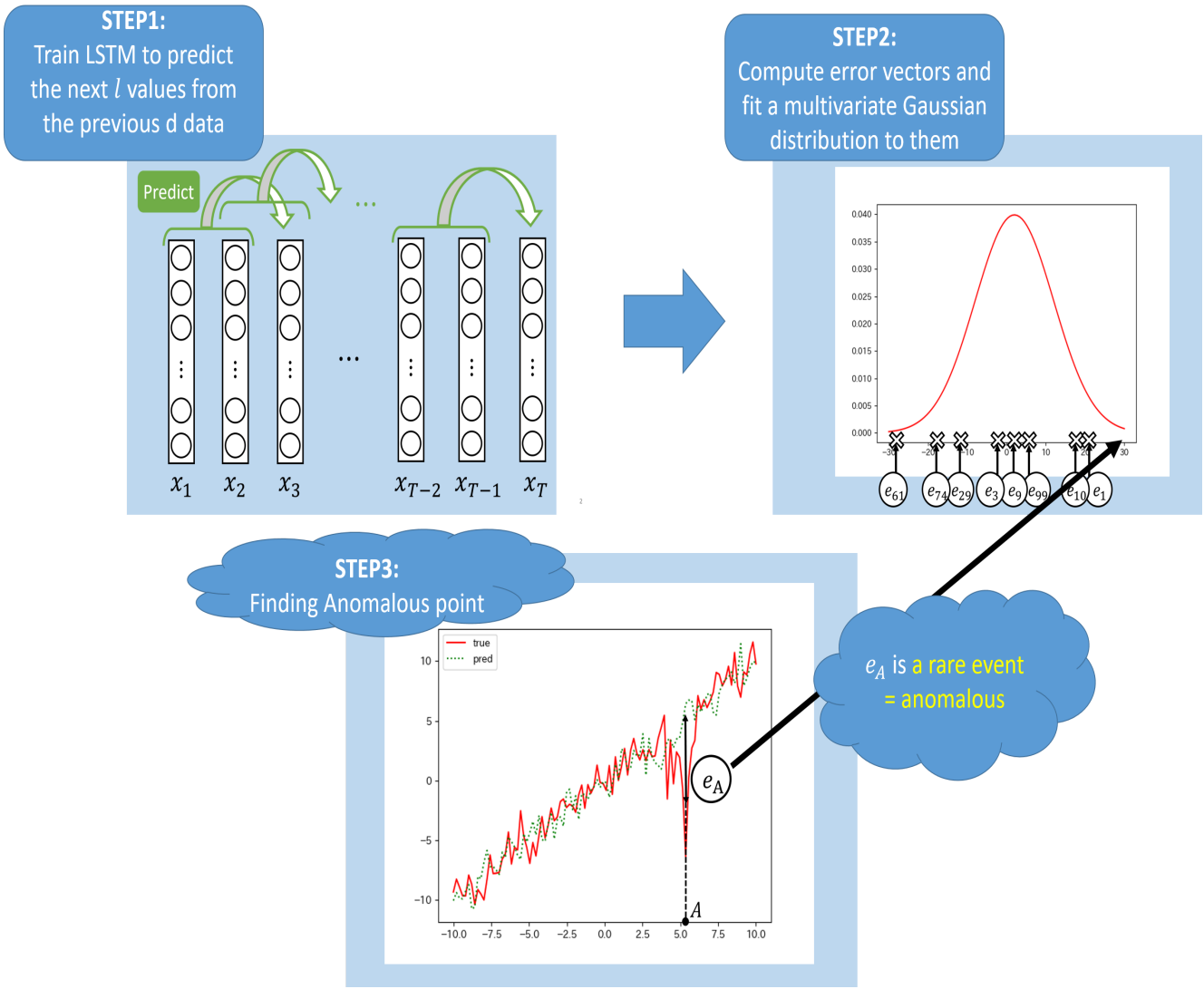
$$e = x_{\text{true}} - x_{\text{pred}}$$

using the trained LSTM. $x_{\text{true}}, x_{\text{pred}}$ are the observed value and the predicted value respectively. Then, we fit a multivariate Gaussian distribution to error vectors computed over test data by the maximum likelihood estimation.

STEP3

Compute the error vector at the point where an anomaly is likely to have happened. If that vector is located at the end of the Gaussian distribution estimated in STEP2, conclude an anomaly happened.
Suppose we want to detect an anomaly at point A in the figure below. When the error vector is located at the end of the distribution, something anomalous was likely to have happened at that point. We can assume

Rare error vector occurred ⇒ Probability distribution of data has changed from normal to anomalous



Mahalanobis' Distance

As discussed in the previous chapter, we can measure the rarity of the event with the location in the distribution. The Mahalanobis' distance is statistics representing an anomaly score. Assuming the parameters of an M dimensional Gaussian distribution are estimated as follows

$$p(x|\mathrm{Data}) = N(x|\hat{\mu}, \widehat{\sum}).$$

Then, the Mahalanobis' distance is defined

$$a(x) = (x - \hat{\mu})^{\top} \widehat{\sum}^{-1} (x - \hat{\mu}).$$

We can measure the rarity of the event with $a(x)$.

Required Libraries

- matplotlib 2.2.2

- pandas 0.23.1
- numpy 1.14.5
- scikit-learn 0.19.1
- scipy 1.1.0

In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from copy import deepcopy
from sklearn.preprocessing import StandardScaler

import renom as rm
from renom.optimizer import Adam
from renom.cuda import set_cuda_active
set_cuda_active(False)
```

Loading Data

We will use ECG dataset, qtdb/sel102 ECG dataset [2].

It can be downloaded from

http://www.cs.ucr.edu/~eamonn/discords/

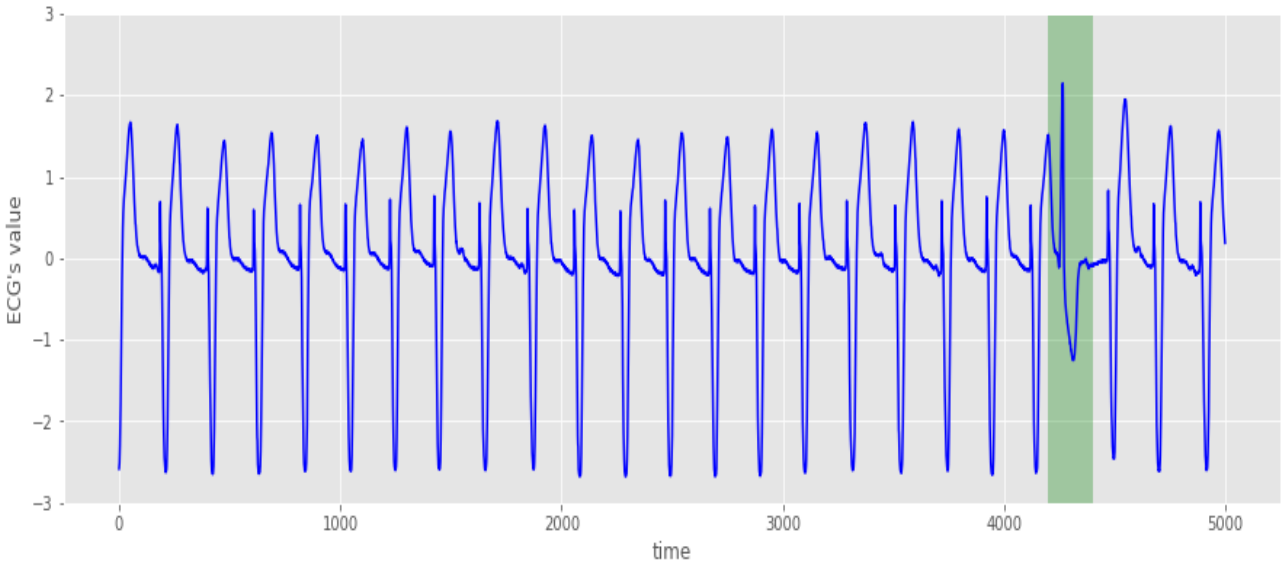
Preprocessing

Firstly, we will standardize ECG data and plot a part of it (0~5000 time) to see the structure.

In [2]:
```
df = pd.read_csv('data/qtdbsel102.txt', header=None, delimiter='\t')
ecg = df.iloc[:,2].values
ecg = ecg.reshape(len(ecg), -1)
print('length of ECG data : ', len(ecg))

# standardize
scaler = StandardScaler()
std_ecg = scaler.fit_transform(ecg)

plt.style.use('ggplot')
plt.figure(figsize=(15,5))
plt.xlabel('time')
plt.ylabel('ECG\'s value')
plt.plot(np.arange(5000), std_ecg[:5000], color='b')
plt.ylim(-3, 3)
x = np.arange(4200,4400)
y1 = [-3]*len(x)
```

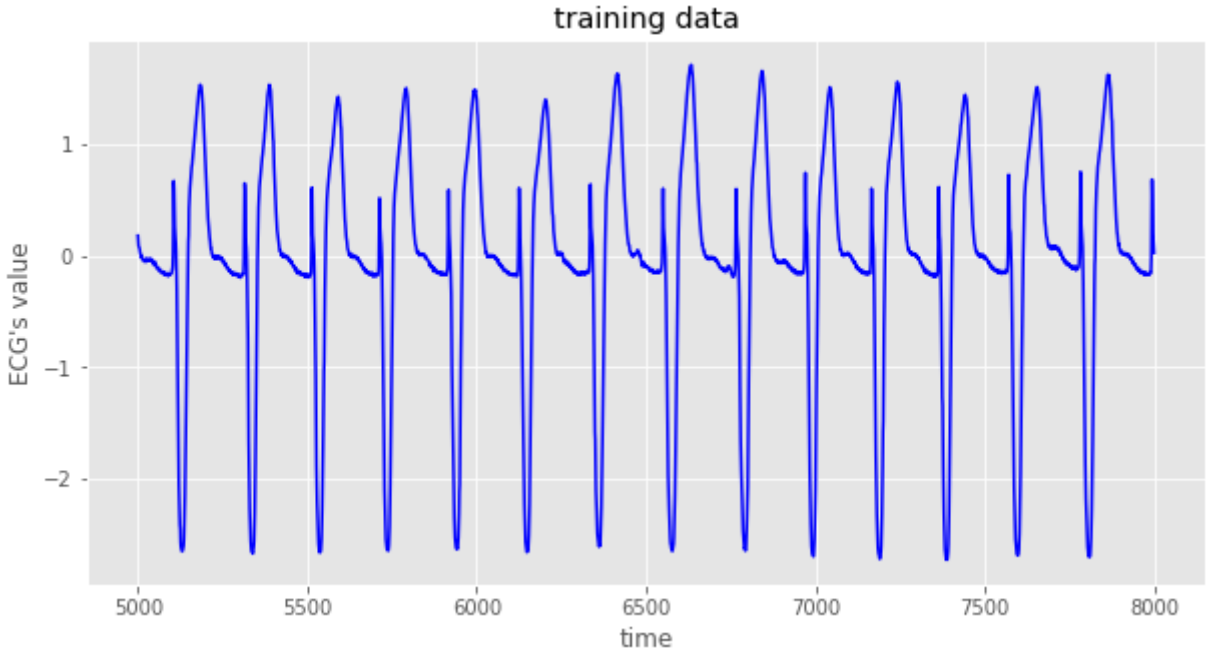
```
y2 = [3]*len(x)
plt.fill_between(x, y1, y2, facecolor='g', alpha=.3)
plt.show()
```

```
length of ECG data :  45000
```



As shown in the graph above, this data is periodic. We can also observe that there is a change point at around time 4250 because the periodicity is collapsing at around there. Since the aim of STEP1 is to get LSTM learned from normal data, we will use data after time 5000 as training data.

In [3]:
```
normal_cycle = std_ecg[5000:]

plt.figure(figsize=(10,5))
plt.title("training data")
plt.xlabel('time')
plt.ylabel('ECG\'s value')
plt.plot(np.arange(5000,8000), normal_cycle[:3000], color='b')# stop plot a
plt.show()
```



Secondly, we will define the function which creates sets of a subsequence of d length and a label of l dimension as follows.

In [4]:
```
# create data of the "look_back" length from time-series, "ts"
# and the next "pred_length" values as labels
def create_subseq(ts, look_back, pred_length):
    sub_seq, next_values = [], []
    for i in range(len(ts)-look_back-pred_length):
        sub_seq.append(ts[i:i+look_back])
        next_values.append(ts[i+look_back:i+look_back+pred_length].T[0])
    return sub_seq, next_values
```

In this tutorial, we will set $d = 10, l = 3$.

In [5]:
```
look_back = 10
pred_length = 3

sub_seq, next_values = create_subseq(normal_cycle, look_back, pred_length)


X_train, X_test, y_train, y_test = train_test_split(
    sub_seq, next_values, test_size=0.2)

X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)


train_size = X_train.shape[0]
test_size = X_test.shape[0]
print('train size:{}, test size:{}'.format(train_size, test_size))
```

```
train size:31989, test size:7998
```

Model Definition

In [6]:
```
# model definition
model = rm.Sequential([
    rm.Lstm(35),
    rm.Relu(),
    rm.Lstm(35),
    rm.Relu(),
    rm.Dense(pred_length)
    ])
```

Parameters Setting

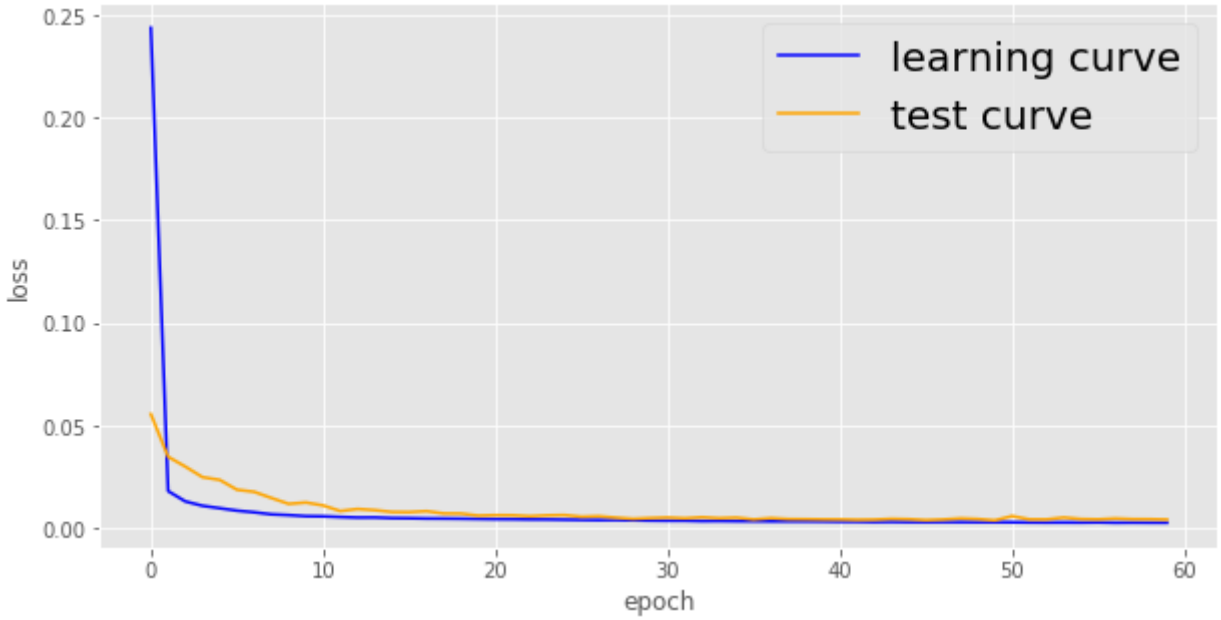
In [7]:
```
# params
batch_size = 100
max_epoch = 2000
period = 10 # early stopping checking period
optimizer = Adam()
```

Train Loop（STEP1）

In [8]:
```
# Train Loop
epoch = 0
loss_prev = np.inf

learning_curve, test_curve = [], []

while(epoch < max_epoch):
    epoch += 1

    perm = np.random.permutation(train_size)
    train_loss = 0

    for i in range(train_size // batch_size):
        batch_x = X_train[perm[i*batch_size:(i+1)*batch_size]]
        batch_y = y_train[perm[i*batch_size:(i+1)*batch_size]]

        # Forward propagation
        l = 0
        z = 0
        with model.train():
            for t in range(look_back):
```

```
                z = model(batch_x[:,t])
                l = rm.mse(z, batch_y)
            model.truncate()
        l.grad().update(optimizer)
        train_loss += l.as_ndarray()

    train_loss /= (train_size // batch_size)
    learning_curve.append(train_loss)

    # test
    l = 0
    z = 0
    for t in range(look_back):
        z = model(X_test[:,t])
        l = rm.mse(z, y_test)
    model.truncate()
    test_loss = l.as_ndarray()
    test_curve.append(test_loss)

    # check early stopping
    if epoch % period == 0:
        print('epoch:{} train loss:{} test loss:{}'.format(epoch, train_los
        if test_loss > loss_prev*0.99:
            print('Stop learning')
            break
        else:
            loss_prev = deepcopy(test_loss)

plt.figure(figsize=(10,5))
plt.plot(learning_curve, color='b', label='learning curve')
plt.plot(test_curve, color='orange', label='test curve')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(fontsize=20)
plt.show()
```

```
epoch:10 train loss:0.005752936793859102 test loss:0.012398829683661461
epoch:20 train loss:0.004399360207910869 test loss:0.006011407356709242
epoch:30 train loss:0.0036237839880168764 test loss:0.004803447052836418
epoch:40 train loss:0.003143804723347266 test loss:0.004181258846074343
epoch:50 train loss:0.002822675645682774 test loss:0.003704755799844861
epoch:60 train loss:0.0025706934453598386 test loss:0.004091349896043539
Stop learning
```



As we can see in the figure above, the learning of LSTM has converged. Next, we go to STEP2, fitting an M dimensional Gaussian distribution to error vectors.

Fitting an M dimensional Gaussian distribution（STEP2）

It's well known that the maximum likelihood estimator of a Gaussian distribution can be computed as

$$\hat{\mu} = \frac{1}{N}\sum_{n=1}^{N} x^{(n)}$$

$$\hat{\Sigma} = \frac{1}{N}\sum_{n=1}^{N}(x^{(n)} - \hat{\mu})(x^{(n)} - \hat{\mu})^{\top}$$

In [9]:
```
# computing errors
for t in range(look_back):
    pred = model(X_test[:,t])
model.truncate()
errors = y_test - pred

mean = sum(errors)/len(errors)

cov = 0
for e in errors:
    cov += np.dot((e-mean).reshape(len(e), 1), (e-mean).reshape(1, len(e)))
cov /= len(errors)

print('mean : ', mean)
print('cov : ', cov)
```

```
mean :  [-0.00471252  0.00561184  0.01125641]
cov :  [[0.00093565 0.00088413 0.00097755]
 [0.00088413 0.00208558 0.0025572 ]
 [0.00097755 0.0025572  0.00498106]]
```

Anomaly Detection (STEP3)

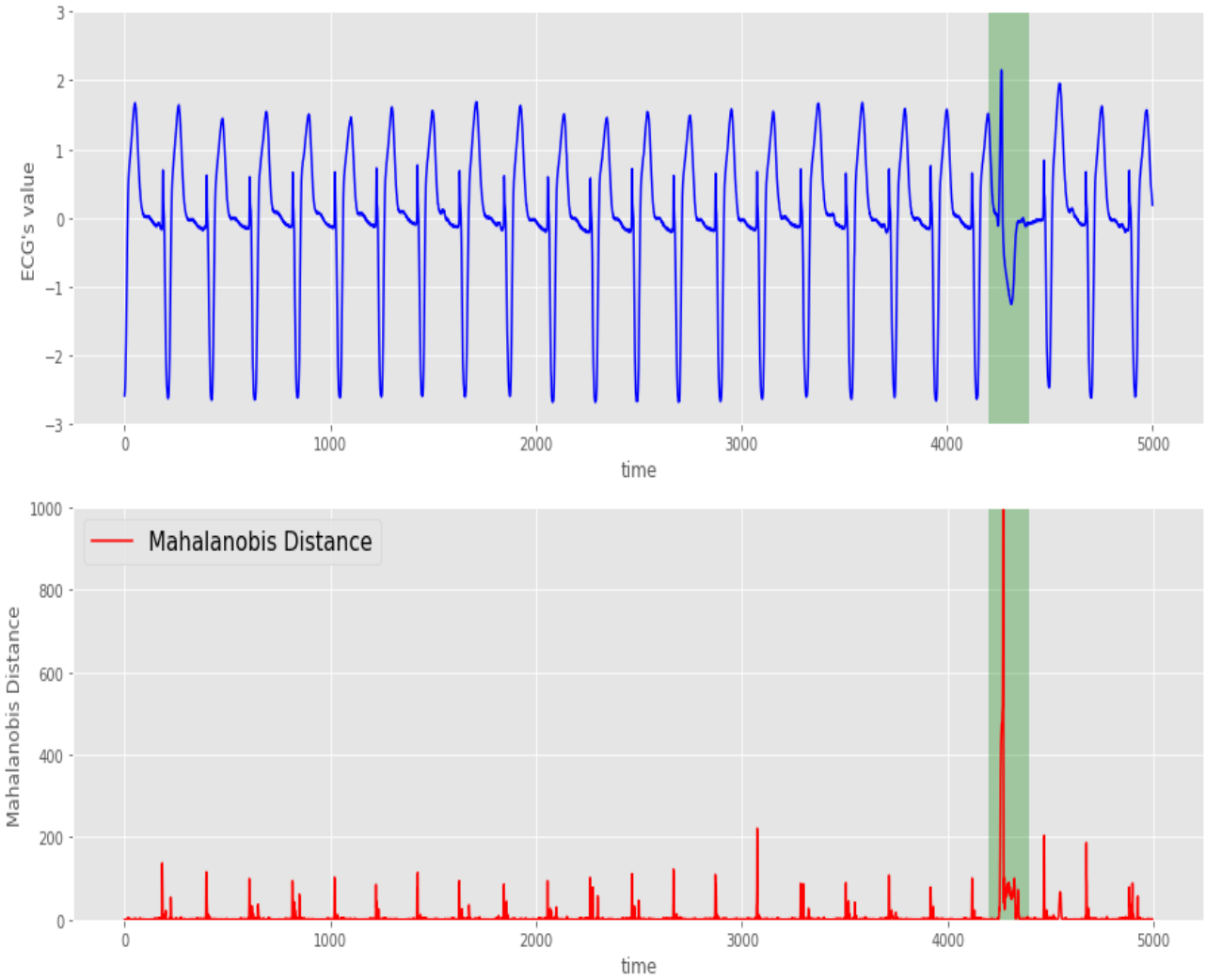
We will verify if this algorithm works even for unknown data.
Firstly, we will create sets of a subsequence and a label. Then we will compute error vectors in the same manner.

In [10]:
```
# calculate Mahalanobis distance
def Mahala_distantce(x,mean,cov):
    d = np.dot(x-mean,np.linalg.inv(cov))
    d = np.dot(d, (x-mean).T)
    return d

# anomaly detection
sub_seq, next_values = create_subseq(std_ecg[:5000], look_back, pred_lengt
sub_seq = np.array(sub_seq)
next_values = np.array(next_values)

for t in range(look_back):
    pred = model(sub_seq[:,t])
model.truncate()
errors = next_values - pred
```

Secondly, we will plot the Mahalanobis' distance for each error vector and corresponding ECG data.

In [11]:
```
m_dist = [0]*look_back
for e in errors:
    m_dist.append(Mahala_distantce(e,mean,cov))

fig, axes = plt.subplots(nrows=2, figsize=(15,10))

axes[0].plot(std_ecg[:5000],color='b',label='original data')
axes[0].set_xlabel('time')
axes[0].set_ylabel('ECG\'s value' )
axes[0].set_ylim(-3, 3)
x = np.arange(4200,4400)
y1 = [-3]*len(x)
y2 = [3]*len(x)
axes[0].fill_between(x, y1, y2, facecolor='g', alpha=.3)

axes[1].plot(m_dist, color='r',label='Mahalanobis Distance')
axes[1].set_xlabel('time')
axes[1].set_ylabel('Mahalanobis Distance')
```

```
axes[1].set_ylim(0, 1000)
y1 = [0]*len(x)
y2 = [1000]*len(x)
axes[1].fill_between(x, y1, y2, facecolor='g', alpha=.3)

plt.legend(fontsize=15)
plt.show()
```



As we can see in the figure above, the Mahalanobis' distance (a measure of rarity) got large at around time 4250 compared to other parts. As we saw in the preprocessing part, this data has the periodicity collapsing point at around time 4250. As a result, we were able to detect the anomaly.

Conclusion

We explained the anomaly detection algorithm for time series data using LSTM. As shown in the experiment part, we were able to find the anomaly by rarity, in other words, the location in the distribution.

References

[1] Malhotra, Pankaj, et al. "Long short term memory networks for anomaly detection in time series." Proceedings. Presses universitaires de Louvain, 2015.

[2]E. Keogh, J. Lin and A. Fu (2005). HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence. In The Fifth IEEE International Conference on Data Mining