

You have **2** free stories left this month.  
Sign up and get an extra one for free.

# Anomaly Detection in Time Series Data Using Keras



Tek Raj Awasthi

Jul 29 · 10 min read ★

Anomaly Detection in time series data provides e-commerce companies, finances the insight about the past and future of data to find actionable signals in the data that takes the form of anomalies.

## Introduction

In this project, we'll build a model for Anomaly Detection in Time Series data using Deep Learning in Keras with Python code. you must be familiar with Deep Learning which is a sub-field of Machine Learning. Specifically, we'll be designing and training an LSTM Autoencoder using Keras API, and Tensorflow2 as back-end. Along with this you will

also create interactive charts and plots with plotly python and seaborn for data visualization and displaying results within Jupyter Notebook.

## **What is Time Series Data?**

Time Series is a sequence of numerical data collected at different points in time in successive order. This is not a cross-sectional data. This is an observation on the value of a variable at different times.

Time series data can be found in business, science, finance. Few examples of time series data are Birth rates, GDP, CPI(Consumer Price Index), Blood Pressure tracking, Global Temperature, population, insights on a product.

Time Series data are very important for prediction. These data are used for understanding past outcomes, predicting future outcomes, making progress strategies, and more.

## **What is Anomaly Detection in Time Series Data?**

Anomaly Detection in the data mining field is the identification of the data of a variable or events that do not follow a certain pattern. Anomaly detection helps to identify the unexpected behavior of the data with time so that businesses, companies can make strategies to overcome the situation. It also helps the firms to detect the error and frauds that are going to happen at particular time, or it helps

to learn from past histories of data that showed unusual behavior.

Applying machine learning in anomaly detection helps to increase the speed of execution. Machine learning algorithm's implementation helps the companies to find simple and effective approaches for detecting the anomalies. Since machine learning algorithms are able to learn from data and make predictions so applying these algorithms in anomaly detection of time series data carries huge impact on its performance. There are various applications of anomaly detection in time series data in different domain topics.

## **What is LSTM ?**

LSTM stands for Long Short-term Memory, which is also an artificial neural network similar to Recurrent Neural Network(RNN). It processes the data passing on the information as it propagates. It has a cell, allows the neural network to keep or forget the information. Here, I have just introduced about LSTM for your ease. If you want to know more about it, you can search it in google.

## **Implementation**

### **Datasets**

As a case study we are gonna be working with S&P 500 Index to detect and predict anomalies. By anomalies I mean sudden price change in S&P index.

## **What is S&P 500 index ?**

S&P 500 is a stock market index that tracks the stock performances of top 500 large-cap US companies listed in stock exchanges. This index represents the performances of stock market by reporting the risks and reporting of the biggest companies. The people in the finance industry consider it as one of the best stock market index in US.

In this project, we'll work with this data , but captured from 1986 and 2018. This data was stored and collected on kaggle and I have downloaded it locally in my desktop.

## **Implementation**

We will be using Python and also designing deep learning model in keras API for Anomaly Detection in Time Series Data. You need to be familiar with TensorFlow and keras and understanding of how Neural Networks work.

### **Step 1: Importing the libraries**

Here, we will be using TensorFlow, NumPy, pandas, matplotlib, seaborn and plotly libraries from python.

`% matplotlib inline` sets the background of matplotlib to inline because of which the output of plotting commands will be displayed *inline* within frontends like the Jupyter notebook, directly below the code cell.

```
import numpy as np
import tensorflow as tf
import pandas as pd
pd.options.mode.chained_assignment = None
import seaborn as sns
from matplotlib.pylab import rcParams
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
%matplotlib inline
sns.set(style='whitegrid', palette='muted')
rcParams['figure.figsize'] = 14, 8
np.random.seed(1) tf.random.set_seed(1)
print('Tensorflow version:', tf.__version__)
```

## Step 2: Loading S&P 500 Index Data

Download the dataset from here: [Click here](#)

In the time series data graph, Dates(year) are in X-axis and the closing price on the Y-axis.

Now we'll read the dataset which is CSV file, using `pd.read_csv` where we have imported pandas as `pd`. And the leftmost column parses the states into pandas date-time

format. And let's view few rows at the top using head() function.

```
df = pd.read_csv('S&P_500_Index_Data.csv',  
parse_dates=['date']) df.head()
```

Output:

	date	close
0	1986-01-02	209.59
1	1986-01-03	210.88
2	1986-01-06	210.65
3	1986-01-07	213.80
4	1986-01-08	207.97

Now checking the shape of our dataset, which will show (8192, 2) i.e. 8192 entries and 2 columns.

```
df.shape
```

Now, let's see the closing price of the stock from 1986 to 2018. Here we have used plotly, and we'll use a sub-module graph\_objects from plotly . Now we will populate the figure using add\_trace() method which helps to plot different

types of charts in the same figure. And Scatter mode is set to 'line' plot. Legend value is set to 'close' which is closing stock value and then update the figure layout.

```
fig = go.Figure()  
fig.add_trace(go.Scatter(x=df.date,  
y=df.close, mode='lines', name='close'))  
fig.update_layout(showlegend=True)  
fig.show()
```

Output:



You will see the date and closing stock value when you hover your mouse over the plot. Now here comes the Anomaly detection into play to tell you when should you buy or shell the stock as it shows the outlier in the data.

## Task 3: Data Preprocessing

Data preprocessing is a very important task in any data mining process as the raw data may be unclear, it may be missing the attributes, it may contain noise, wrong or duplicate data.

Here, we are going to standardizing our target vector by removing the mean and scaling it to unit variance. Before standardization, let's split the dataset into training and testing set. We have taken 80% of data frame for training and remaining 20% for testing. And then `iloc` method will allocate the data from index 0 to `train_size` to train set and remaining to test set.

```
train_size = int(len(df) * 0.8)
test_size = len(df) - train_size
train,
test = df.iloc[0:train_size],
df.iloc[train_size:len(df)]
print(train.shape, test.shape)
```

Then you can see the inline output as (6553, 2) (1639, 2) as the size of training and test set respectively. Now, let's create the instance of `StandarsScaler` function and then fit this helper function on the training set and then transform the train and test set.



```
from sklearn.preprocessing import
StandardScaler
scaler = StandardScaler()
scaler = scaler.fit(train[['close']])
train['close'] =
scaler.transform(train[['close']])
test['close'] =
scaler.transform(test[['close']])
```

Now data standardization task is performed here.

## Task 4: Create Training and Test Splits

Since this is time series data, we need to create the subsequences before we go to using the data to train our model. We can create sequences with a specific time step, it's 30 in our case. That means we need to create the sequences with 30 days for the historical data.

And as required by LSTM network, we need to reshape our input data into shape and sample by n time\_steps by n features. In our case, the n is equal to 1 i.e. one feature.

```
def create_dataset(X, y, time_steps=1):
    Xs, ys = [], []
    for i in range(len(X) - time_steps):
        v = X.iloc[i:(i + time_steps)].values
        Xs.append(v)
        ys.append(y.iloc[i + time_steps])
    return np.array(Xs), np.array(ys)
```

Here, we have just converted the list into numpy arrays, where data are from  $i$  to  $i + \text{time\_steps}$  are located to X array and remaining to Y array.

```
time_steps = 30
X_train, y_train =
create_dataset(train[['close']], train.close,
time_steps)
X_test, y_test =
create_dataset(test[['close']], test.close,
time_steps) print(X_train.shape)
```

Now it prints the output as (6523, 30, 1) i.e. 6523 entries are taken for training, time\_steps as 30 and 1 feature.

## Task 5: Build an LSTM Autoencoder

In this step, we are gonna build an LSTM Autoencoder network and visualize the architecture and data flow. So here's how we are going to detect anomalies using an autoencoder.

For that , first, we need to train the data with no anomalies and then take the new data point and try to reconstruct that using an autoencoder.

If the reconstruction error for the new dataset is above some threshold, we are going to label that example/data point as an anomaly.

In the following 2 lines, we have just assigned values from X\_train array i.e. (6523, 30, 1)

```
timesteps = X_train.shape[1]
num_features = X_train.shape[2]

from tensorflow.keras.models import
Sequential
from tensorflow.keras.layers import Dense,
LSTM, Dropout, RepeatVector, TimeDistributed
model = Sequential([ LSTM(128, input_shape=
(timesteps, num_features)), Dropout(0.2),
RepeatVector(timesteps), LSTM(128,
return_sequences=True), Dropout(0.2),
TimeDistributed(Dense(num_features)) ])
model.compile(loss='mae', optimizer='adam')
model.summary()
```

Here, we have used the Sequential model from Keras API. Our sample data is 1% which is 2D array and is passed to LSTM as input. The output of the layer is going to be a feature vector of input data.

We have created one LSTM layer with the number of cells to be 128. Input shape is equal to no. of time\_steps divided by no. of features. Then we have added the Dropout regularization to 0.2. Since our network is LSTM, we need to duplicate this vector using RepeatVector. It's purpose is to just replicate the feature vector from the output of LSTM layer 30 times. Our encoder is done here.

## Decoder Layer

Now we have mirrored the encoder in reverse fashion i.e. decoder. TimeDistributed function creates a dense layer with number of nodes equal to the number of features. And the model is compiled finally using adam optimizer function which is gradient descent optimizer. And the model summary is shown as follow:

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 128)	66560
dropout (Dropout)	(None, 128)	0
repeat_vector (RepeatVector)	(None, 30, 128)	0
lstm_1 (LSTM)	(None, 30, 128)	131584
dropout_1 (Dropout)	(None, 30, 128)	0
time_distributed (TimeDistri	(None, 30, 1)	129
Total params: 198,273		
Trainable params: 198,273		
Non-trainable params: 0		

### Summary Of Model

## Task 6: Train the Autoencoder

Now, let's create Keras callback and use EarlyStopping so that we don't need to hard code the number of epochs.

If our network doesn't improve for 3 consecutive epochs, i.e. validation loss is not decreased we are going to stop our training process. That is the meaning of patience.

And now let's fit the model to our training data. No. of epochs is set to high as higher the epochs, more the accuracy of training. 10 % of the data is set for validation. And then the callback is done using es i.e. EarlyStopping.

```
es =  
tf.keras.callbacks.EarlyStopping(monitor='val_  
_loss', patience=3, mode='min')  
history = model.fit( X_train, y_train,  
epochs=100, batch_size=32,  
validation_split=0.1, callbacks = [es],  
shuffle=False )
```

## Task 7: Plot Metrics and Evaluate the Model

Now we'll plot the matrix that is training loss and validation loss using matplotlib.

```
plt.plot(history.history['loss'],  
label='Training Loss')  
plt.plot(history.history['val_loss'],  
label='Validation Loss') plt.legend();
```

In our plot, validation loss is consistently found to be lower than training loss that means the training data due to the high dropout value we used So you can change the hyperparameters in 5th step to optimize the model.

We need to still predict the anomaly in our test data by calculating the mean absolute error on the training data. First, let's get prediction on our training data. And then we evaluate the model on our test data.

```
X_train_pred = model.predict(X_train)
train_mae_loss =
pd.DataFrame(np.mean(np.abs(X_train_pred -
X_train), axis=1), columns=['Error'])

model.evaluate(X_test, y_test)
```

Then distribution loss of training mean absolute error is shown using seaborn.

```
sns.distplot(train_mae_loss, bins=50,
kde=True);
```

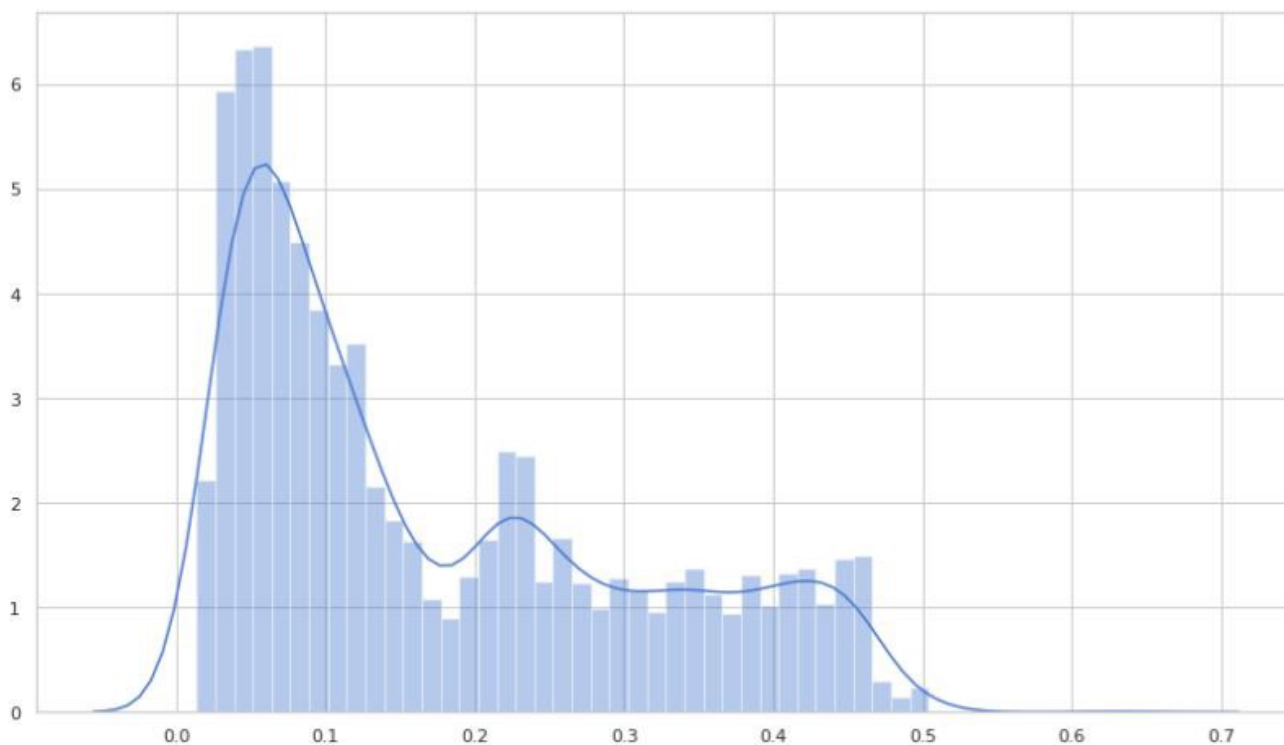
That shows the output like:

Here, we can set the threshold as 0.65 as no value is larger than that.

Now, let's calculate the mean absolute error on test set in similar way to the training set and then plot the distribution loss.

```
X_test_pred = model.predict(X_test)
test_mae_loss = np.mean(np.abs(X_test_pred -
X_test), axis=1)

sns.distplot(test_mae_loss, bins=50,
kde=True);
```



## Task 8: Detect Anomalies in the S&P 500 Index Data

Now we are going to build a data frame containing loss and anomalies values. Then let's create a boolean-valued column called an anomaly, to track whether the input in

that corresponding row is an anomaly or not using the condition that the loss is greater than the threshold or not. Lastly, we will track the closing price

```
THRESHOLD = 0.65
test_score_df =
pd.DataFrame(test[time_steps:])
test_score_df['loss'] = test_mae_loss
test_score_df['threshold'] = THRESHOLD
test_score_df['anomaly'] = test_score_df.loss
> test_score_df.threshold
test_score_df['close'] =
test[time_steps:].close
```

Now, let's see the first five entry of our dataframe

```
test_score_df.head()
```

and for last 5 entries

```
test_score_df.tail()
```

which shows result like this,

	date	close	loss	threshold	anomaly
8187	2018-06-25	4.493228	0.638251	0.65	False



8188	2018-06-26	4.507583	0.691009	0.65	True
8189	2018-06-27	4.451431	0.696043	0.65	True
8190	2018-06-28	4.491406	0.726802	0.65	True
8191	2018-06-29	4.496343	0.708928	0.65	True


## Final Result

Now let's plot train and test loss value and overlay the line for threshold. First, we will create an empty figure and then use `add_trace()` method to populate the figure. We are going to create line plot using `go.Scatter()` method

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=test[time_steps:].
date, y=test_score_df.loss, mode='lines',
name='Test Loss'))
fig.add_trace(go.Scatter(x=test[time_steps:].
date, y=test_score_df.threshold,
mode='lines', name='Threshold'))
fig.update_layout(showlegend=True) fig.show()
```

This shows result like this





The plot looks like we are thresholding extreme values quite well. All the values above the horizontal orange line are classified as Anomalies. That's it.

Thank you for reading.

The complete source code link to my **GitHub** : Click Here

**You can reach me at:**

<https://www.linkedin.com/in/tekrajawasthi34456b162/>

<https://twitter.com/dotpyarmy>

<https://www.facebook.com/debuglife>

• • •

*Originally published at <https://valueml.com> on July 29, 2020.*

---

## Sign up for Top Stories

By The Startup

A newsletter that delivers The Startup's most popular stories to your inbox once a month. [Take a look](#)

Your email

---

Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

Anomaly Detection

Time Series Analysis

Time Series Forecasting

Keras

TensorFlow

[About](#) [Help](#) [Legal](#)

Get the Medium app

