Time Series – part II
Implementation

Applied Mathematics

Shay Malkin
Tal Ladijensky
Nir Titelbom

Common methods frequently used in the field of time series forecasting

There are 12 different classical time series forecasting methods commonly used in forecasting time series:

1. Autoregression (AR)
2. Moving Average (MA)
3. Autoregressive Moving Average (ARMA)
4. Autoregressive Integrated Moving Average (ARIMA)
5. Seasonal Autoregressive Integrated Moving-Average (SARIMA)
6. Seasonal Autoregressive Integrated Moving-Average with Exogenous Regressors (SARIMAX)
7. Vector Autoregression (VAR)
8. Vector Autoregression Moving-Average (VARMA)
9. Vector Autoregression Moving-Average with Exogenous Regressors (VARMAX)
10. Simple Exponential Smoothing (SES)
11. Holt Winter's Exponential Smoothing (HWES)
12. Time-Delay Neural Network (TDNN)

How can we model our time series?

Prophet is a forecasting tool designed for analyzing time-series that display patterns on different time scales such as yearly, weekly and daily.
The TDNN forecasting works in most cases for stationary data for forecasting short time periods. Prophet is a way to crack the cases which TDNN can't handle by building a better tool to deal with these cases. Prophet handled these cases better by using knowledge from Bayesian statistics, including seasonality, the inclusion of domain knowledge, and confidence intervals to forecast data by better estimating better the risk with the data given.

Visualization of the time-series data

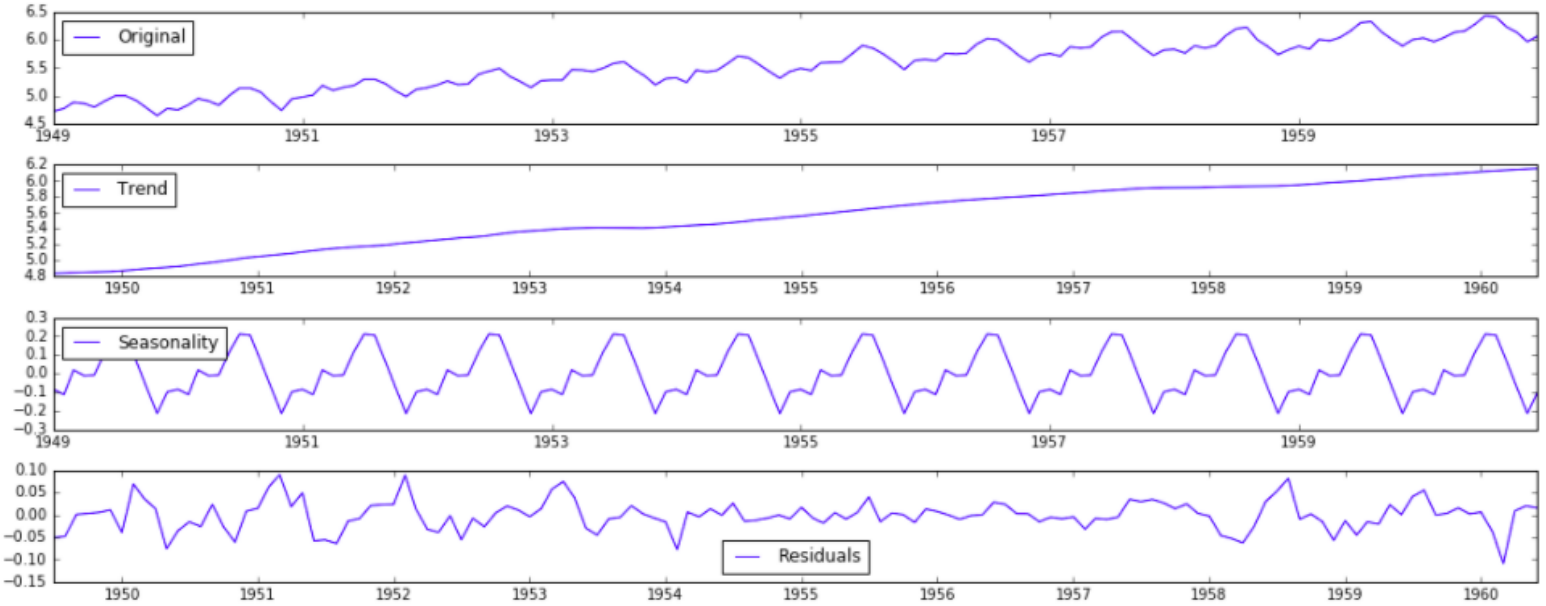
We can use a tool called time-series decomposition that allows us to decompose our time series into the three components we talked about in the last presentation: trend, seasonality, and noise. An example in python:

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(ts_log)

trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.subplot(411)
plt.plot(ts_log, label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
```

The result of previous example:



Stationary

Most Time-Series forecasting models assume that the series is stationary. How can we make our Time-Series Stationary?

There are several methods that can work well in some cases and others will work on well on different cases while the other sill not achieve the result wanted. We will explain the 2 methods used in most cases.

1. Differencing

In short, we are given an n samples: $\mathrm{X} = \{x_1, x_2, x_3, x_4, \dots, x_n\}$. We assume this time series is not stationary. We create a new time series by taking the difference between the data points. The time series with difference of degree 1 becomes: $\tilde{X} = \{x_2 - x_1, x_3 - x_2, \dots., x_n - x_{n-1}\}$.

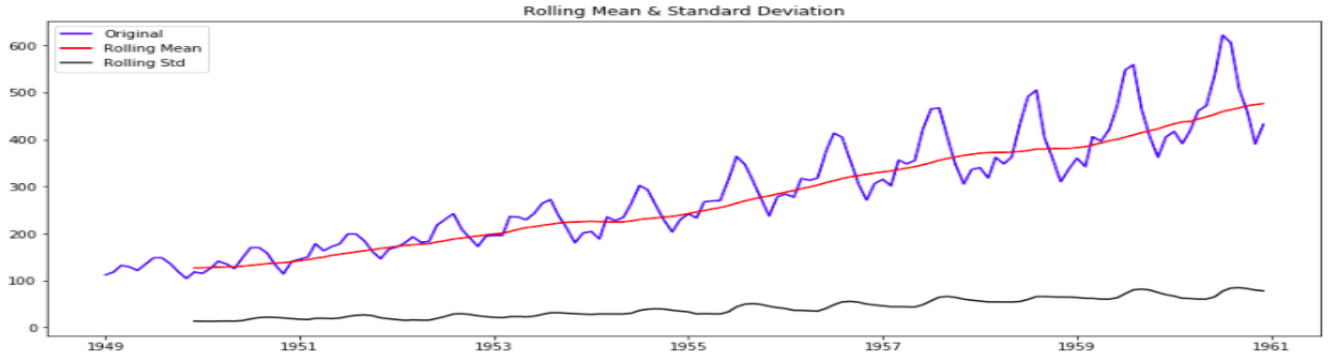
We will then check if there is any improvement in X autocorrelation.
If not, we can try a second, a third or even higher order differencing.
We need to be careful with this method because the more we difference, the analysis and all of the calculations and conclusions are going to be harder to execute.

2. Transformation

In many time series the variance is not constant and it is not stable. Transformation is a way normally suggested only when Differencing is not working.
The most common transformation is the log transform , other known transformations include power transform and square root transformation.

An example in python:

Lets assume that out series is not stationary.



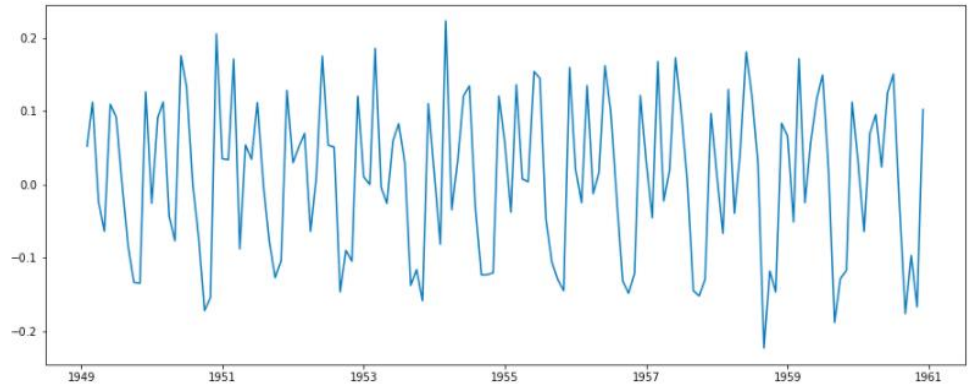

```
Results of Dickey-Fuller Test:
Test Statistic                  0.815369
p-value                         0.991880
#Lags Used                     13.000000
Number of Observations Used    130.000000
Critical Value (5%)             -2.884042
Critical Value (1%)             -3.481682
Critical Value (10%)            -2.578770
dtype: float64
```

This series is not stationary because the mean values are increasing while the standard deviation is small throughout the time series data.

We will transform the series with log transformation and we will take the difference of the value at a particular time with that of the previous time by 1.

```
#Take first difference:
ts_log_diff = ts_log - ts_log.shift()
plt.plot(ts_log_diff)
```

```
[<matplotlib.lines.Line2D at 0x11a74da50>]
```



We can see here that we achieved a stationary time series to penetrate the model, in order in the end to get the wanted result we will simply apply the Inverse function.

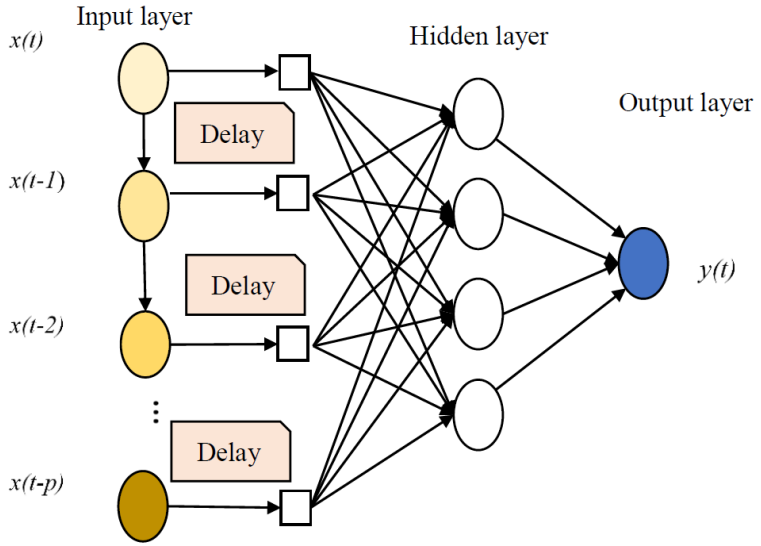
Time-Delay Neural Network (TDNN)

TDNNs can be referred to as feedforward neural networks, except that the input weight has a delay element associated with it. The time series data are often used in the input and the finite responses of the network can be captured. Accordingly, a TDNN can be considered as an ANN architecture whose main purpose is to work on sequential data.

As we know, In the feed forward network (FNN), the input nodes are the successive observations of the time series, i.e. the target $x(t+1)$ is a function of the values $x(t-i), i = 0,1,2,\dots,p$ where p is the number of input nodes.

In TDNN, the input nodes are the time series values at some particular lags. A TDNN has multiple layers and scient inter-connection between units in each layer to ensure the ability to learn complex nonlinear decision surfaces.

The next figure shows the architecture of a TDNN. The structure of the TDNN includes an input layer, one or more hidden layers, and an output layer. Each layer contains one or more nodes determined through a trial and error process of the given data, as there is no theoretical basis. In here, the figure present one hidden layer in the TDNN's structure.



In addition, as shown, the network input layer utilized the delay components embedded between the amounts of input-units to attain the time-delay. Each node had its own values and through the network computation to achieve the output results. Under the input–output relationship function of the network, if the output result is the next time prediction of the input x, there must be a certain relationship between present and future.

This is given by $y = x(t+1) = H[x(t), x(t-1), \ldots, x(t-p)]$. Consequently, the TDNN is to seek the activation function H of the input–output in the network. This is given by $\Phi_j = \sum_{i=0}^{p} w_{ij} x(t-i) + \theta_j$ and $y(t) = \sum_j w_{jk} \boldsymbol{\Psi}(\Phi_j)$, where Φ_j and $y(t)$ are the function at the input and output layers, respectively; p is the number of tapped delay nodes. w_{ij} is the weight of the i^{th} neurons in the input layer into the j^{th} neurons in the hidden layer; and θ_j is the bias weight of the j^{th} neurons.

The activation function $\boldsymbol{\Psi}$ represent ReLU or a nonlinear sigmoid.

We define the function as: $sigmoid \stackrel{\text{def}}{=} \frac{1}{1+e^{-x}}$; $ReLU \stackrel{\text{def}}{=} \max(0, x)$.

The precise architecture of TDNNs (time-delays, number of layers) is mostly determined depending on the classification problem and the most useful context sizes. The delays or context windows are chosen specific to each application.

In time delay networks , the connections have time delays of different length. Such a time delay postpones the forwarding of a unit's activation to another unit. When using multiple time delays, these networks can be trained to deal with a sequence of past time series elements. At each time step, a single sequence element is fed into the input, but the network's forecast takes preceding sequence elements into account.

One of the common uses is in detecting temporal patterns for stock market prediction tasks.

Another model - CNN

Convolutional Neural Network is a feed-forward neural network. Like the traditional architecture of a neural network including input layers, hidden layers and output layers, convolutional neural network also contains these features and the input of the layer of convolution are the output of the previous layer of convolution or pooling. Of course, they still have some unique features such as pooling layers, full connection layers, etc.

The computation of convolution in a CNN model usually accepts a 2D image as its input. But, if our goal is predict the stock price movement, that belongs to1D time series data, it is necessary to change the function to help us do the computation. The new function we adopt in the model accepts a number of stock data, and other properties such as the number of the filters, width of the filters and stride as its input.

The outputs of our function are a number of matrices, and the specific number is decided by the number of the filters. CNN model will extract some features from them and they also will be the input of the next pooling layer after the computation of activation function.

The input form can be defined as following matrix: $X = \left(x^{(1)}, x^{(2)}, \ldots, x^{(m)}\right)$

Here m indicates the number of samples as inputs. Each sample consist of k features, so the input of the neural network is a $m \times k$ matrix.

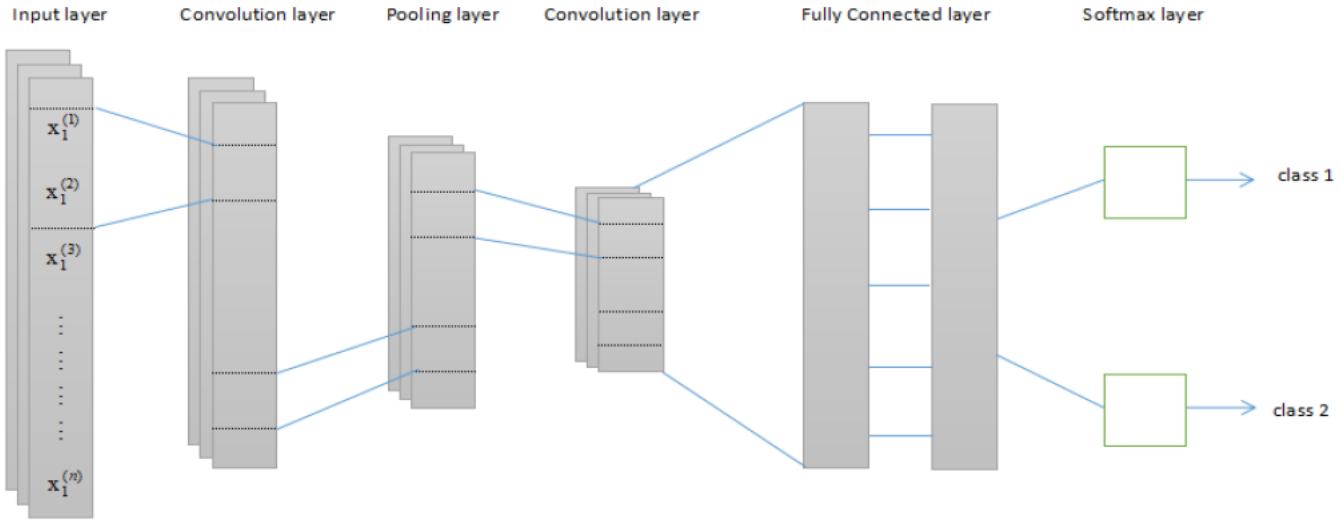
For getting the final classification values, which will tell us the stock will be up or down in future, we will use "teaching" number to train the model.

The output form is basically similar as the input form: $Y = \left(y^{(1)}, y^{(2)}, \ldots, y^{(m)}\right)$

Here each y is a Boolean value: 1 or 0. Additionally, the input data should be normalized because there is too much difference between each feature.

Traditionally, the activation function that people always use can be concluded as followings: Tanh, Relu, Sigmoid and Lrelu. To solve the problem of gradient vanishing, we choose the Relu and Lrelu function to do the computation of activation.

An example the architecture of CNN model:



Deep learning libraries

TensorFlow

TensorFlow is the most widely used framework for deep learning.

Created by the Google Brain team, TensorFlow is an open source library for numerical computation and large-scale machine learning. It uses Python to provide a convenient front-end API for building applications with the framework, while executing those applications in high-performance C++. TensorFlow allows the user to build static computational graphs, that is, a graph structure is built once, and then execute the computation with different values. TensorFlow also computes the gradients of the graph, making backpropagation and thus, training a model, very easy.

TensorFlow is optimized to run efficiently on CPU, GPU and TPU (Tensor Processing Unit, a piece of hardware produced by google to fit this purpose exactly).

Keras

Although TensorFlow is a flexible and powerful tool, it can be somewhat difficult to use. The user needs to hard code a lot of details in the model.

It takes time, effort and knowledge to use TensorFlow.

That where Keras comes in!

Keras is an API built on top of TensorFlow designed specifically to enable the user to build and modify neural networks. It is very useful for research, when one wishes to try many different model variations, Keras makes it very easy.

In general, Keras makes deep learning applications very accessible to everyone.

Nowadays, TensorFlow comes with Keras built in.

PyTorch

Another widely used library for deep learning is PyTorch.

PyTorch was developed by Facebook and is open sourced and based on the Torch library. It has a very friendly syntax, very similar to Numpy.

The main difference between PyTorch and TensorFlow is that PyTorch uses dynamic computational graphs as opposed to static ones.

If static graphs are built once, a dynamic graph is built over and over in every iteration of the training. On one hand, this is less efficient, having to build the same structure multiple times. On the other hand, this allows changing the graph during runtime (if some criteria is met, for example).

Other libraries for deep learning exist. For example, Caffe2 (Facebook), MXNet (Amazon), CNTK (Microsoft) and more.

The End!