timeline: A Time Series Visualization Platform

Manuel Haid, Thomas Mauerhofer, and Matthias Wölbitsch

Knowledge Technologies Institute

Abstract. In this paper a web-based visualization platform for time series data is proposed. It is not only able to represent static temporal data and it properties but also data from live sources, such as sensors. Furthermore, it allows the visualization of the output of time series prediction methods. The main aspects of this paper are the overall design, the use cases, and the outlook for possible enhancements of the application.

Keywords: Time Series, Visualization, Forecasting

1 Introduction

In today's society data plays a import role in many different ways. For instance, businesses are using large amount of data to predict consumer behavior to satisfy their demand, or rely on it for operations entirely. Often time plays a essential role in these real-world data sets and applications. The following list contains three representative examples in no particular order:

1. The consumption of electricity does not only depend on the time of the day, but also on the day of the week and other seasonal effects. It is important to model the future consumption to avoid failures of large portions or the complete power grid.
2. In the area of weather forecasting time is of the essence. Many applications (e. g., shipping) require precise and accurate forecasts to operate on minimal risk.
3. In the finance market many decisions are made within small time spans and are based on the prediction of complex models. Incorrect or late data can lead to significant financial losses.

This kind of data sets with an temporal importance are usually referred to as time series (i. e., two- or higher-dimensional data with time as one dimension). A more formal definition [6] of a time series $\{Y\}_t$ is a time ordered collection of observations $(y_1, y_2, \ldots, y_n)$. This implies two trivial consequences. First, the samples in the time series are ordered, and second, previous observations may influence future values but never the other way around. A element of the time series recorded at time or period t, denoted as y_t, can either be a scalar value or a vector. Time series of scalars are denoted as *univariate* times series, whereas a time series of higher dimensional data is called a *multivariate* time series. For this project we only focused on univariate time series data and refer to to them simply as time series in the context of this paper.

When dealing with time series usually one of the first steps in the analysis process is the visualization of the data. The graphical representation can already give some great insights in properties and characteristics of the data. For example, a simple visual inspection can show patterns, trends or seasonal effects without the use of advanced statistical methods. These properties are important since they have to be considered in the modeling of the time series. One example for a statistical forecasting model that requires knowledge of the seasonal and stationary properties of a data set is the Box–Jenkins [5] model.

The goal of our project is it to create a web-based platform that allows the user to perform this initial visual inspection of the time series in a convenient but sophisticated way. This includes, of course, the possibility to create simple time series plots (i. e., visualizations that plot the data against the time) but also more advanced plots such as the auto-correlation function (ACF) plot. Another goal is the support of a quick and responsive visualization of forecasts on the a data set. This means that is should be possible to quickly inspect the deviation of the prediction from the actual test data including a confidence band.

The rest of the paper is structured as follows. Section 2 contains a detailed description of our platform, used algorithms and overall design. Section 3 contains the conclusion of the project and a discussion of possible future work and improvements.

2 Solution

In the first step of our design process was the task to decide what types of data sets should be supported by our web-based platform. We decided that the following types are sufficient for the most time series analysis tasks:

1. A simple time series data set. This consist mainly of a time indexed collection of scalar data points and meta data. The meta data is mostly

made up of some descriptive information (i. e., the name of the data set and a description) and some other details about the data which can be directly derived from it (e. g., the number of samples, the period,. . .).

2. A specialized version of the simple time series data set were the data is not static but can be updated. We denote this type of data set as live time series data set. The meta data of the set is basically the same as for the simple time series data set, however, it provides special methods that allows the appending of new data samples at the end of the time series. This type of data set is especially useful when dealing with data from live sources (e. g., sensors).
3. A data set which can be used to investigate time series forecasts. This object contains the same descriptive meta information as the simple time series data set as well, however, it differs in the actual data it can hold. It basically separates a time series into multiple partitions. First, the training data that was used to train the forecasting algorithm and perhaps a time stamp which indices what portion of the training data was used as validation set to determine the performance during the training. Furthermore, the prediction (i. e., the forecasted time series) is the second part of the data set. Optionally, there is should be the possibility to include the test data (i. e., the actual values that should be predicted).

After the types of the data sets were fixed, one of the most important design decisions for the project had to be made. It was how the `timeline` platform should be used by the users. Our first idea was that the platform should be a standalone application. However, in our opinion, this causes some major usability and implementation issues. First, there is no nice and easy way to import the three different types data sets at the startup time from files. It would be very hard to specify a data format that can be utilized by all three data set types. For example, most of the real-world data sets are usually stored in the form form of `.csv` files. These typically do not allow the storage of additional meta information, at least not without special parsing rules. Furthermore, the live data sets would require a description of the live data source (e. g., a socket) in some way and the forecasting data sets would either require multiple data files (i. e., one for the training set, test set, and the prediction) or one file with multiple disjoint sections.

Another possibility for the data import in the standalone application use case would be the possibility to import the data using a dialog in the web-interface. However, since data sets can be quite large (i. e., multiple thousand samples), the upload time could be substantive. Moreover, since it was not planned to support persistent storage of the data sets in a rational or document-based database (at least not for the first release), the usability of the platform would suffer due to repeated re-uploads of the data sets.

Therefore, we decided that the best way to use `timeline` would be as library. This allows the user a much more flexible usage of the platform. We anticipate more or less the following workflow with `timeline` and designed the application such that it can be used in that way:

1. The user works on a time series analysis or time series forecasting problem using Python or a Jupyter notebook.
2. The user wants to visualize some aspects of his or her work and imports the `timeline` library.
3. He or she creates one or more `timeline` data set objects that corresponds to the problem he or she is working on and registers it within the library.
4. In the last step he or she can start the `timeline` web-service on a specific port and can visit the web-interface to view all available visualizations and other information.

The usage of the platform as library also allows a simpler integration of live data sources by simply wrapping the data steam into a class that implements all required live data set methods and the possibility to simply include data sets from other web resources (e. g., from a website that publishes stock market data). Altogether we think that this design choice supports a more natural and quicker workflow with the platform.

Additionally, there are also some example data sets in the library available. It is possible to generate random data sets with 1000 data points that are sampled from a normal distribution with a random mean in the set $\{0, 1, \dots, 10\}$ and a random standard deviation in the range $[0, 2]$. Furthermore, there is a internet traffic data set from the Time Series Data Library [8] available. There is a live data set that generates one new sample per second from a normal distribution with some random mean and standard deviation in the same ranges as before. And there is a simple forecasting data set available which is also based on the internet traffic data set.

The visualization primitives that can be generated by the platform are the following:

1. A simple time series plot, which plots the observations (i.e. the data points) against the time of the observations. It is possible to filter

the time series in a certain time span and include a rolling mean and rolling standard deviation into this plot. This can be used to determine whether or not the time series is stationary [10].

2. A extended version of the simple time series plot that allows the plotting of multiple time series plots in one figure to compare them directly.
3. A extended version of the simple time series plot that works in the conjunction with live data sets such that it can be updated dynamically.
4. A autocorrelation plot [5], also known as autocorrelation function (ACF) plot, which are a statistical tool to proof seasonal effects in the data.
5. A forecast evaluation plot, that can be used to inspect the performance a time series prediction method. It is a extended version of the simple time series plot as well, which shows the multiple parts of the forecast data set in one plot.

The back-end was implemented in Python using Flask [2] as web-framework to render the user interface, deliver data using the API, and provide a small web-server such that the application can be executed. To store the time series data in the back-end the data analysis library Pandas [9] was used. The application user interface uses the responsive Bootstrap [1] front-end framework and the JavaScript visualization library MetricsGraphics.js [3] to render the visualization primitives.

3 Conclusion and Outlook

Like in most other projects, there is a lot room for potential improvements and possible feature work. One constraint, due to the implementation of the framework, is that it only can handle dates and time stamps in a resolution of seconds. This is caused by the fact that all date and time related parameters are encoded using the UNIX time stamp (i. e., the number of seconds since midnight January 1, 1970 UTC) before they are sent from the front-end to the back-end and vice versa. This is required since dates are often part of URLs, for example, in API requests. An implication of this is, of course, that it is only possible to handle time series data within a resolution of one second (e. g., the period between two samples must be at least one second) in the `timeline` platform. This could be problematic, especially for real-time applications, where events usually happen in the range of milliseconds. However, there are potential solutions for this issue. For example, one option would be to encode dates as milliseconds since January 1, 1970 or a more general, and probable better, solution would be the usage of a well-defined date time format that allows the reliable parsing of dates from strings, which could be transmitted instead of the UINIX time stamp integers.

Another possibility for feature work would be the creation of a Python package that can be uploaded to PyPI [4], the Python package index. This would require to write a installer using the Python `setuptools` module. Furthermore, the number of available visualizations is quite small at the time, and the existing plots could be more interactive (e. g., brushing). There are plenty of other plots that could be integrated into the platform, for example, seasonality plots. However, not only additional visualizations could be added to the platform, but also algorithms and statistical tests that allow further examination of the data sets (e. g., implementation of the Dickey–Fuller test [7]).

In conclusion we think that the `timeline` visualization platform could be a valuable tool when doing time series analysis. It could easily integrated in existing workflows and is modular structured to allow enhancements. There is room for potential improvements, but the first version of this application lays a solid foundation for future use cases.

References

1. Bootstrap front-end framework. http://getbootstrap.com/, accessed: 2016-06-14
2. Flask micro web framework. http://flask.pocoo.org/, accessed: 2016-06-14
3. Metricsgraphics.js. http://metricsgraphicsjs.org/, accessed: 2016-06-15
4. Pypi – the python package index (python software repository). https://pypi.python.org/pypi, accessed: 2016-06-15
5. Box, G.E., Jenkins, G.M.: Time series analysis: forecasting and control, revised ed. Holden-Day (1976)
6. Cortez, P., Rio, M., Rocha, M., Sousa, P.: Multi-scale internet traffic forecasting using neural networks and time series methods. Expert Systems 29(2), 143–155 (2012)
7. Dickey, D.A., Fuller, W.A.: Distribution of the estimators for autoregressive time series with a unit root. Journal of the American Statistical Association 74(366a), 427–431 (1979)
8. Hyndman, R.: Time series data library. http://data.is/TSDLdemo/, accessed: 2016-06-14
9. McKinney, W.: Data structures for statistical computing in python. In: van der Walt, S., Millman, J. (eds.) Proceedings of the 9th Python in Science Conference. pp. 51 – 56 (2010)
10. Nason, G.: Stationary and non-stationary time series (2006)