Time Series Analysis Support for Data Scientists Software Requirements Specification

E. Petersen (ezekielp) - 2-8-2021 - v1.5

Table of Contents

1. SRS Revision History

This lists every modification to the document. Entries are ordered chronologically.

Date	Author	Description
1-22-2021	ezekielp	Added initial document to repo, will modify template for project
1-23-2021	ezekielp	Replaced template text with project specifics in ConOps 2 - 2.5
1-28-2021	ezekielp	Replaced template text with project specifics in ConOps 2.6
2-1-2021	ezekielp	Finished section 3 (except 3.1 and 3.2), finished v1 of SRS
2-6-2021	ezekielp	Minor edits, still waiting on final API to finish 3.1/3.2
2-8-2021	ezekielp	Added 3.1/3.2 according to my interpretation of the template

2. The Concept of Operations (ConOps)

The concept of operations (ConOps) "describes system characteristics for a proposed system from the users' viewpoint." (IEEE Std 1362-1998) The ConOps document communicates overall system characteristics to all stakeholders.

2.1. Current System or Situation

Data Scientists are uniquely situated to apply emerging scientific and machine learning processes and algorithms to analyze real world data and make important inferences about the future. One such category of inferences concerns Time Series forecasting, a task for predicting information about a wide range of human activity to improve real world decision making. For example, predicting solar irradiance as a function of time can help solar engineers make operational decisions at renewable energy plants. Predictions made by programs created by Data Scientists can be vital to maximizing energy capture and minimizing operating costs.

There are existing libraries, such as those provided by pandas or scikit-learn that Data Scientists currently use to analyze various data sets and generate "pipelines" to convert raw TS data into usable forecasting models for industry professionals and others. Data Scientists use these libraries to decide which pipeline most accurately predicts the future state of the TS. However, this is a non-linear search for effective pipelines that is currently performed in a largely "test and check" manner.

2.2. Justification for a New System

The current "system" is not as thorough or efficient as it could be because Data Scientists are forced to do non-linear searches for effective pipelines manually. In a literal sense, there is no existing system for achieving a comprehensive search for pipelines at low cost to the Data Scientist.

To illustrate the problem, we introduce the math involved for constructing a hypothetical pipeline. Even in a modest case where we apply 3 preprocessing directives out of 10 available to us, a single modeling strategy out of 2, and a single evaluation strategy out of 3, there are thousands of possible pipelines that could be formed (10 * 9 * 8 * 2 * 3 = 4,320), which would be difficult to track and compare without additional software support. Since we know that our list of basic transformations described later (which approximately reflects the above situation) is not exhaustive and we do not include the various plotting or graphical analysis steps available in the hypothetical pipeline, we can reasonably assume this is by no means the worst-case problem complexity. So, given the shortcomings of the current solutions, we argue that our system can improve the accuracy of these pipelines and the productivity of the Data Scientists to develop better forecasting models.

2.3. Operational Features of the Proposed System

Our new system aims to represent the challenge of conducting a non-linear search for an effective pipeline and model by treating the transformation steps taken by Data Scientists as nodes in a "Transformation Tree". We aim to give Data Scientists a library of functions to modify this Transformation Tree and keep all possible pipeline steps in one place to be later executed and compared. These possible modifications will include creating a new tree, inserting new operator nodes, replacing existing operators with another, replicating subtrees or tree paths, and adding a subtree or path to an existing node. We also wish to allow Data Scientists to extract successful pipelines or trees and save/load them when necessary to put into production or update/maintain later.

By organizing the TS transformation steps as an n-ary tree, Data Scientists will be able to keep all design decisions in one place and compare the many variants of a pipeline. However, the system will limit types of operators to follow the logical progression from preprocessing to modeling and finally to evaluation to ensure data consistency as it moves through the tree. This will prevent nonsensical situations such as an attempt to evaluate a model prior to a model being created. In order to achieve this, nodes and operators are classified according to their role in the pipeline process.

2.4. User Classes

For the proposed system, there is only one major user class: Data Scientists. We are defining a library of functions and tree transformations to be used by these Data Scientists to streamline their everyday processes. We are assuming a proficient level of programming knowledge such that users may call our transformation functions to process, analyze, and visualize a given Time Series in ways they see fit according to their broad domain knowledge. Data Scientists already use the libraries behind the functional modules of our system, so it is reasonable to assume Data Scientists will understand and be able to use a synthesis and repackaging of those libraries.

2.5. Modes of Operation

As there is only one user class and because the end result will be a library for use by Data Scientists, the library itself is the only true "mode of operation". There will be unit tests and

example scripts for using the library, however, primary use will be by Data Scientists for use in their own scripts and analysis.

2.6. Operational Scenarios (Also Known as "Use Cases")

Use Case 1: Compare model performance of multiple pipelines

Brief description: This use case describes how a Data Scientist would quickly compare the effectiveness of several different TS pipeline options via a Transformation Tree.

Actors: A Data Scientist

Pre-conditions:

1. The Data Scientist has access to a csv file of a Time Series.
2. The Time Series may be represented with or without timestamps to go along with the magnitudes. The Data Scientist knows this file formatting detail.
3. The Data Scientist has access to a machine with reasonable computing power (8 or more GB of RAM and >30GB of hard drive space).

Steps to Complete the Task:

1. The Data Scientist gains access to the csv file containing TS data and places it in the data folder reachable by the program.
2. The Data Scientist figures out:
    (a) How many preprocessing steps are relevant to test on the given dataset. Perhaps there is missing data and they wish to impute the data on all possible pipelines. They may wish to also try standardizing the data in some pipelines but not others.
    (b) Which ML modeling algorithms they wish to test (there are MLP and RF options that may be added to the pipelines).
    (c) How they wish to test accuracy. There are several algorithms for comparing forecasted data with test data.
3. The Data Scientist creates a Transformation Tree, which by default will handle the initial read of the TS file on execution.
4. The Data Scientist adds nodes as needed according to the needs of step 2 using the library of tree functions available.
5. The Data Scientist "executes" the tree using another provided tree function and providing the file path of the input csv.

Post-conditions:

The user has a printed list of pipeline accuracy measurements associated with the steps (nodes) contained in the pipeline. They will be able to identify the "best" pipeline they tested.

Use Case 2: Alter an existing Transformation Tree

Brief description: This use case describes how a Data Scientist would alter a tree they have tested to use different operators or add operators to improve performance.

Actors: A Data Scientist

Pre-conditions:

1. The Data Scientist has a script creating a Transformation Tree

2. The Data Scientist determines that there is no pipeline tested with acceptable accuracy.

Steps to Complete the Task:

1. The Data Scientist figures out which steps are most likely to be unhelpful or which steps could be useful when added to the Transformation Tree.
2. The Data Scientist uses more of the provided library functions to edit the operator of one or mode nodes in the tree according to their evaluation of performance.
3. The Data Scientist re-executes the Transformation Tree.

Post-conditions:

The user has another printed list of pipeline accuracy measurements associated with the steps (nodes) contained in the pipeline. They will be able to identify the "best" pipeline they tested.

Use Case 3: Enter a useful pipeline into production

Brief description: This use case describes how a Data Scientist can save a working pipeline and apply it to other Time Series'.

Actors: A Data Scientist

Pre-conditions:

1. The Data Scientist has post-condition information from Use Case 1 or Use Case 2.
2. The Data Scientist can evaluate acceptable accuracy from the pipeline information.
3. There are additional TS csv files that need to be forecasted.

Steps to Complete the Task:

1. The Data Scientist identifies the steps (nodes) that were present in the desired pipeline.
2. The Data Scientist passes the pipeline information to a provided library function which will save that sub-tree or path to an external file.
3. The Data Scientist loads the subtree or path using another provided function, which will generate a pipeline that produces the desired preprocessing steps and desired model.
4. The Data Scientist will input the new TS files to the pipeline.

Post-conditions:

The Data Scientist has forecasting information about the Time Series in question and can make pass it on to the direct stakeholders (solar plant engineers for example) to make further operational planning decisions based on that information.

3. Specific Requirements

3.1. External Interfaces (Inputs and Outputs)

There are three primary external interfaces to the system, largely following the use cases described above, but with some minor differences. These are a subset of the functions described in the User Guide of the README.

3.1.1. Testing Execution

1. Purpose: To execute a full Transformation Tree and compare the accuracy of a set of pipelines.
2. Source of input/destination of output: Input will be a single csv file accessible by the Tree. Output will be printed to standard output of accuracies of each pipeline of the tree. Side effects will be saved forecasts in the form of csvs.
3. Valid range of inputs and outputs: Inputs must be a valid string filename. Outputs will be one or more csv files and printed accuracies between 0 and 1.
4. Units of measure: Accuracies will be decimal values.
5. Data formats: Data formats will be csv files and numeric values.

3.1.2. Pipeline Execution

1. Purpose: To generate actionable forecast information in the form of a csv.
2. Source of input/destination of output: Input will be a single csv file accessible by the Tree. Output will be a saved csv forecast.
3. Valid range of inputs and outputs: Inputs must be a valid string filename. Outputs will be one csv file.
4. Units of measure: N/A
5. Data formats: Data formats will be csv files.

3.1.3. Saving/Loading

1. Purpose: To save a Tree object in an external file to be used later or perhaps moved. Additionally, to reconstruct that object from file by loading the Tree object.
2. Source of input/destination of output: Input will be a Tree object and the output will be a pickle file in the case of a save. In the case of a load, the input will be the filename of the pickle, and the output will be a Tree object. Pickles are stored in the pickle_objects directory.
3. Valid range of inputs and outputs: The input will only ever be a single filename and/or a single Tree object. The output will only ever be a pickle file or a Tree object.
4. Units of measure: N/A
5. Data formats: As described above, files will be pickles, and objects to be saved/loaded will be Tree objects.

3.2. Functions

1. Validity checks on the inputs: All tree modifications are expected to leave the tree in a state with valid node ordering. That is, nodes representing later steps in a pipeline may not come before steps that must take place first. When executing, the Tree must check that an appropriate node is at the root to allow for the proper preprocessing.
2. Sequence of operations in processing inputs: When given a file, the system will convert the file to a useable internal data structure, modify it according to the directives given to it by the Data Scientist, and then use the internal data structures to produce a model with a forecast.

3. Responses to abnormal situations: When the user attempts to illegally modify the Tree, the system should raise an exception. Similarly, if the system cannot access a particular file when trying to execute or load a tree, it should also raise an exception.

3.3. Usability Requirements

The software should be able to model and evaluate TS data as reliably as the existing libraries in use by Data Scientists for manual pipelining. Since ML is notoriously complex and random sampling during the modeling process will change the results from execution to execution, we cannot guarantee any level of accuracy beyond the accuracy of the resulting models. We are merely constructing the framework to use to test models.

The comparison of the many pipelines that can be present in a tree must be presented in a readable and easily comparable format so it does not take an extraordinary amount of time to understand the output of the system.

3.4. Performance Requirements

The software accepts one input file (a .csv) at a time and will not be able to process another until completion in both testing and production modes. Future updates to the system could allow for sequences of input files in production mode.

The time in which it takes for the software to complete is largely dependent on the complexity of the tree structure created by the Data Scientist, on the models used, and by the size of the input file. A single pipeline on a file with less than 1000 rows of data should be processed in less than a few minutes.

3.5. Software System Attributes

The software must be **maintainable** in the sense that new functions can be added to the existing module libraries without major changes to the structure of the nodes or tree. Our system will not have an exhaustive list of every modification to a Time Series that a Data Scientist may want to apply. By factoring out the logic, we allow for simple addition to the tables of functions to be used in nodes when new functions are written.

The software must be able to check that a tree is in the proper form prior to any modification to it and certainly prior to execution. This will primarily be done during the “add node” and “add subtree” functions. This will ensure **reliability** of output to conform to the expected behavior. It is important that the software is reliable, as it is merely supposed to improve the existing situation for Data Scientists. If they must troubleshoot or spend more time figuring out why their system is not working, then the time gained from rapid pipeline performance comparisons is lost and the system is moot.

The software should be **portable**, as the saved trees and pipelines should be able to be exchanged among Data Scientists for use on different machines or for different users.

4. Acknowledgements

The SRS template was provided by Juan Flores, who updated it from Anthony Hornof for CIS 422.