Pipeline Trees - An Auxiliary Tool in the Creation of Time Series Pipelines

Juan J. Flores[1,2], Ronny Fuentes[1], Joseph McLaughlin[1], Kyra Novitzky[1], Stephanie Schofield[1], Alec Springel[1], and Seth Tal[1]

[1] University of Oregon, Eugene, OR, USA.
{jflore10,ronnyf,jmclaugh2,knovitzk,sschofie,aspring6,stal}@uoregon.edu
[2] Universidad Michoacana
Morelia, Mexico.
juan.flores@umich.mx,

Abstract. A data science pipeline represents the concatenation of processing steps in developing data science or machine learning models. This article proposes a framework called Pipeline Trees or Transformation-Trees, which aids a data scientist in developing alternative pipelines. In finding a suitable pipeline to solve a data science problem, a data scientist searches for the best order and parameters of pre-processing steps. This search process can be cumbersome and time-consuming; the search space explodes quickly with the number of possible operations to be included as possible steps in the pipeline (and the parameters those steps require). This article presents TTree (Transformation Tree), a python library that allows a data scientist to deal with various options in producing a pipeline. Data science is a vast field; this article highlights the benefits of utilizing machine learning pipeline trees to process, analyze, model, and forecast time series data. TTree forms a Transformation Tree that allows processing and manipulation of the time series data however the Data Scientist sees fit. This library can simplify and expedite the process of time series analysis.

Keywords: Time Series, Forecasting, Pipelines, Modeling, Visualization, Data Science

1 Introduction

Time series data analysis is an important segment of computer science and applies to almost every industry. Many fields observe and record data over a span of time, and make predictions about how the data will change. Many natural phenomena can be also modeled as a time series and their predictions are useful for many fields of research. Predictions of future outcomes and the underlying process to derive predictions can inform and influence the decision-making process (8). For instance, when generating renewable energy with solar panels, the two most important variables that impact the amount of energy produced by a solar cell are solar irradiance and ambient temperature (15). Each of these

variables can be measured over time to create a time series. In this process, solar power plant managers and engineers are interested in estimating how much energy the plant will produce in the near future. Financial time series models assist stockbrokers in forecasting how a stock's price will change, and environmental scientists can record geological data over time to make predictions about earthquakes. The main problem for data scientists is managing the processing, statistical analysis, training, modeling, and visualization of large data sets.

One part of the time series problem is processing "dirty" data into a form that the scientists can work with. Furthermore, large data sets could be riddled with outliers and null values that need to be removed in order to get an accurate picture of the set. After cleaning the data set into a workable form, the data scientist can then begin statistical analysis, training, and modeling their data.

Statistical methods such as ARIMA, Exponential Smoothing, etc., have traditionally solved the forecasting problem (8). Recently, the forecasting problem has attracted the attention of the machine learning community, and there are a variety of methods to produce time series forecasting models. A linear structure that defines a sequence of processing steps to be applied to data is called a pipeline. Pipelines play a similar role in data manipulation as sequential constructs in general programming. The data science community has developed several libraries to implement pipelines; among others, the most prominent of them these days are Scikit-Learn (11) and TensorFlow (3).

Still, there does not exist a mathematical model or a straightforward procedure that takes a data scientist from data to forecasting seamlessly. The data scientist in charge has to explore a pipeline search space, where the set of possible linear arrangements of transformations to form a pipeline can grow quickly. At each step, one can choose several different transformations, in different order, with different operating parameters.

This article presents TTree (Transformation Tree), a python library that allows a data scientist to deal with different options in the production of a pipeline. Data science is an extremely large field; this article focuses and highlights the benefits of utilizing machine learning pipeline trees to process, analyze, model, and forecast time series data. TTree provides the ability to simplify and expedite the process of time series analysis.

2 Pipelines

In data science we often use pipelines, but their fundamental purpose expands to various parts of our daily lives. Despite their many uses, a pipeline's overarching goal is to fetch a data set and perform a sequence of tasks. Eventually, we can observe the output data and pass it as input all over again. More specifically, after data enters a pipeline, various pre-processing functions will transform it to create and train models for diverse purposes. Let us define a pipeline as a sequence of data transformations; Fig. 1 shows one such sequence of transformations.

Let us call X the pipeline input and consider the pipeline consists of transformation steps; F_i denotes the transformation function contained at step i of the

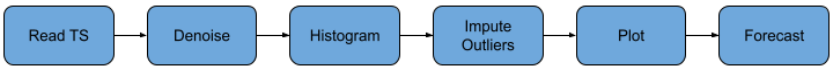

Fig. 1: A Data Transformation Pipeline

pipeline. The pipeline output, Y, can be expressed as the function composition of the transformation functions included in each step, as shown in Eq. (1). We will call each of those functions a pipeline node, and later on, a tree node.

$$Y = F_m(F_{m-1}(...F_1(X)) \tag{1}$$

Most of the data transformation functions require parameters. For instance, in denoising by rolling windows, we need to specify the window width. Eq. 2 includes those parameters, denoted by θ, to each node.

$$Y = F_m^{\theta_m}(F_{m-1}^{\theta_{m-1}}(...F_1^{\theta_1}(X)) \tag{2}$$

One of the most crucial characteristics of a pipeline are their (sequential) order of execution and opportunity for a recurring cycle. Libraries such as Scikit-Learn (11) and TensorFlow (3) have had success in assisting with these issues. Both libraries provide programmers with tools that assist in the use of machine learning pipelines. Compiling neat, coherent, and automated pipelines is an area that no existing framework has conquered. At each step of the construction of a pipeline, a Time Series Data Scientist has various alternatives; different operators can be used to preprocess, model, display, or forecast the time series. Those options must be explored in a systematic way and the experiments that perform such explorations must be properly recorded. Frequently, when those methodological steps are not fulfilled, the Data Scientist ends up walking in circles, repeating experiments that were already performed. Pipeline Trees aid the Data Scientist in that exploration and recording of experiment results. A data tree can be assembled, studied, and when it is ready, it can be executed, freeing the user from having to execute a sequence of experiments related to that exploration of the space of pipelines. At any point of the formation of a TTree, the user can execute any of the pipelines defined in the tree. One of the first processing steps may be to reduce the data size, so that the execution of a pipeline or the whole tree does not take very long. Later on, the user can execute selected pipelines with larger data sizes or with the whole dataset. The TTree library aims to solve many issues around the existing pipeline exploration and organization. To the best of the authors' knowledge, no work covering this topic has been included in any data science or machine learning libraries, nor has it been presented in any conference or journal article before.

3 Pipeline Trees

This section describes the methodology of using tree-like data structures to design and implement Time Series Transformation pipelines. When working with

Time Series data, a Data Scientist wants to transform the data in various ways, i.e., there are alternative methods to stationarize, denoise, model, and forecast. The decisions about what alternatives to take are made iteratively at various points in developing pipelines (those decisions need to be documented and outlined.) Pipeline trees are non-linear data structures for tracking precisely these kinds of non-linear decisions. A Pipeline Tree is called a Transformation Tree TTree, sinc each step in a pipeline performs data transformation operations. Every path from the tree root to a leaf constitutes a pipeline. The functionality provided by TTrees includes testing and executing alternative pipelines in order to compare the results they produce. Once achieved the desired results, pipelines can be extracted directly from a Transformation Tree. Fig. 2 shows an example of a Transformation Tree.

Cayley proposed the tree structure, one of the most studied abstractions in computer science, in 1857 (6). Different programming languages provide diverse implementations of trees. The following definitions provide the basic ground to work with Transformation Trees.

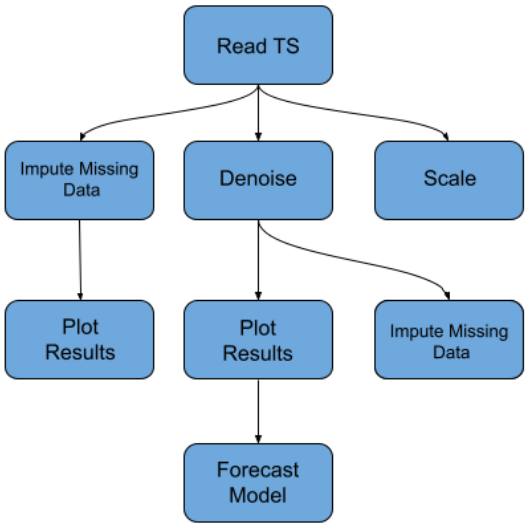

Fig. 2: A Transformation Tree is an n-ary Tree

Definition 1. *Let T be a finite set of data transformation functions, V be a finite nonempty set of vertices, l a total function $l : V \to T$, and E a set of ordered pairs of distinct vertices called edges. $G = (V, l, E)$ is a labeled graph.*

Definition 2. *A labeled tree, T, is a connected, acyclic labeled graph.*

l assigns a transformation function to a tree node. Edges in E are ordered to denote parenthood, i.e., (x, y) means y is x's child.

Definition 3. *The only node that does not appear in the right side of the ordered pairs in E is called the tree root, r, i.e., it does not have a parent.*

Definition 4. *A node that does not appear in the left side of the ordered pairs in E is called a leaf, i.e., it does not have children.*

Definition 5. *A path from r to a leaf is a complete path, p on a transformation tree.*

A complete path describes the transformation process for that given pipeline. Formally, Eq. (2) expresses the transformation that one such path provides to data. The n-ary tree structure defined above is the primary data structure used to organizing a transformation tree, which contains a set of pipelines used on a Time Series.

We can view the Transformation Tree nodes as operators (see Def. 1), with a transformation function associated with them. An operator can provide the necessary pre-processing functionality for Time Series Data, plot the Time Series, or act as a Machine Learning model component. Furthermore, operators performing the same task can use various arguments when needed (see Eq. 2). Pipeline Trees offer a way for the user to track their experimentation process. As a whole, the tree represents a series of alternatives to producing a reasonable forecasting model. At any point during the development of a Tree, the user (a data scientist) can branch away from the current pipeline to test alternatives. Fig. 2 shows an example of how branching on alternatives to a pipeline component produces a Transformation Tree.

TTrees enable us to compare performance between two routes easily since extracting and executing pipelines from these structures is integral in their design. When a set of transformations provides an acceptable level of performance for the task at hand, the user can quickly identify the complete path that offers the most reasonable solution and extract it from the tree. Fig. 3 illustrates the extraction of a pipeline from a Transformation Tree.

This article presents TTrees, a mechanism that provides a Time Series Machine Learning Engineer or Data Scientist with the tools to explore alternatives in the formation of appropriate pipelines, i.e., pipelines that produce Machine Learning Models for Time Series Forecasting with a desired performance level. The functionality provided in TTrees includes the definition of different time series preprocessing, analysis, and forecasting models. Users can define any operations useful to their particular application domain. For instance, many econometric time series are non-stationary; in those cases, the application of differences, logarithms, or other data transformations can be useful. Perhaps, the time series are too noisy and the user wants to augment the TTree with noise removal operations. Perhaps the time series is too simple that an ARIMA model suffices to produce a reasonable forecasting performance, or the data is so chaotic that a sophisticated Deep Learning model is required. Any of those operations can be implemented by a user and used in conjunction with TTree to explore the best pipline for her forecasting needs.

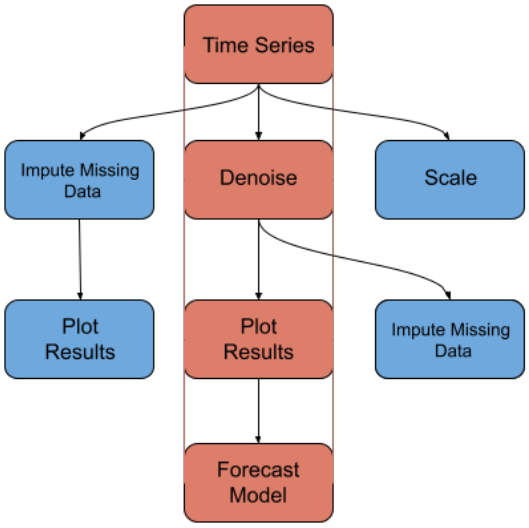

Fig. 3: Pipeline Extraction from a Transformation Tree

4 Implementation and results

The TTree library enables its user to create flexible transformation trees that will cover various use cases. The library's implementation provides three classes: Node, TTree, and Pipeline. The building blocks for constructing a transformation tree are nodes containing the node identifier, a transformation function (operation), and references to its parent and children. Fig. 4 shows the structure of a Transformation Tree node.

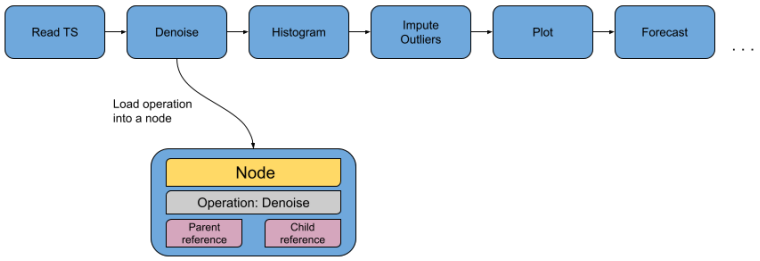

Fig. 4: Transformation Tree node

The available transformation functions can be divided into input/output (`read_csv, store_model`), pre-processing (`denoise, impute_outliers`), statistical analysis (`normality_test, mse, mape`), modeling (`mlp_fit, mlp_forecast`), and visualization (`plot, box_plot, histogram`).

Each transformation function returns a value that becomes the parameter for the next node's function. This flexible Node class allows users to define their own operations, insert them where needed into the transformation tree, and test them by executing the entire tree or specific paths (pipelines).

The TTree class provides methods for creating, modifying, and saving transformation trees. The library also provides functions for deleting, executing, and saving pipelines. The package is open source; the reader can find an exhaustive list of functions available in the package in GitHub.

Every pipeline starts at the root node and ends at a leaf, i.e., it is a complete path (Def. 5). The result of experimenting with a Transformation Tree must be a successful pipeline, i.e., one which produces a reasonable pre-processing, modeling, and forecasting for time series of a given domain. Fig. 5 shows the data flow occurring during the execution of a pipeline extracted from a transformation tree.

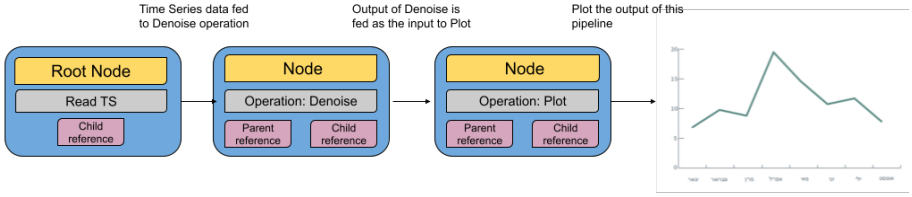

Fig. 5: Data Flow through a Pipeline

One of the functions of the Pipeline class is to extract a selected pipeline, so that it can be tuned and put into production.

A caveat to consider regarding the TTree class is that the user must know function compatibility. The nodes are flexible, accepting any functions they are created with, so users must be cognizant of adjacent nodes' return values and parameters. The return value of a parent node must be compatible with the children's node functions' parameters. This knowledge will ultimately allow the users to alter, remodel, and explore pipelines that can fit their data.

When a tree is created, it contains only a root node, containing a function that loads time series data and returns an array of the data points. A user typically inserts a node containing a plot operator as a root child. A typical path may be a sequence of denoise, impute missing data, and plot transformations; each operation takes an array and returns a modified array. Such a path is attached at the root or its child. After inserting the nodes with their respective operators, the tree can be further modified, executed entirely, or invoked to export desired pipelines. The user can also add a plotting node at the end of each path to demonstrate that particular pipeline's effect. We can see how the pipeline modifies the data at each step; this tracing capability can help find a sequence of operators that create a model that performs reasonably. The transformation tree

also provides the opportunity to modify the model further, exchange operators as needed, and save it for later use.

Fig 6 demonstrates the formation of a TTree using a custom design matrix method and a RandomForest model.

```
>>> root = Node(lambda : read_from_file('temp_historic.csv'))
>>> tree = TTree('Historic Temp Data', root)
>>> tree.add_nodes_byid(0, Node(plot_ts))
>>> tree.add_nodes_byid(0, Node(denoise))
>>> tree.add_nodes_byid(2, Node(impute_missing_data))
>>> tree.add_nodes_byid(2, Node(design_matrix(15, 3))
>>> tree.add_nodes_byid(3, Node(design_matrix(15, 3))
>>> tree.add_nodes_byid(2, Node(design_matrix(15, 3))
>>> tree.add_nodes_byid(3, Node(design_matrix(15, 3))
```

Fig. 6: Formation of a TTree

Several Machine Learning and Deep Learning libraries provide tools to form pipelines, e.g., Scikit-Learn (1) and TensorFlow (2). Data Scientists use pipelines in time series analysis, from preprocessing, statistical analysis, data cleaning, modeling, forecasting, monitoring models performance, etc., and in the final stage, deploying models to production, once those models have reached satisfactory performance levels.

Although those libraries provide tools to implement pipelines, none offers a tool to explore the pipeline space and experiment with different options to form a set of promising pipelines. TTrees is a library that provides such a tool.

5 Related Work

The increasing complexity and size of data processing tasks in time series research, machine learning, and in related interdisciplinary fields has prompted several software projects with the intended goal of improving data processing workflows. This section discusses a collection of related research papers and projects, and how our work contributes to this area.

Data science infrastructure. There is demonstrable need for more robust tools and methods in the field of data science. Several contemporary research papers focus entirely on designing tools to facilitate the operations of data science researchers.

Cingolani et al., (2015) (7) propose a scripting language *(BigDataScript)* to directly address the growing needs of data scientists. BigDataScript is designed to robustly facilitate the construction of data processing pipelines and to provide illustrations of pipeline performance.

Saltz et al., (2019) (13) recognize the growing practical needs of data scientists in their work with by proposing a modified workflow routine derived from

kanban techniques (Anderson (2010) (4)). Their intention is construct a work process that is well defined within a field where the duties and responsibilities are often variable in duration, notably distinct from traditional agile tasks.

Probabilistic models. Researchers in the field of natural language processing have applied similar pipelining methods as a means of performing complex tasks where extracting lexical features from tokens and word groups quickly becomes computationally expensive. Informed by the sucesss of probabilistic models (e.g. Wellner, 2012 (14)), Bunescu, 2008 (5) introduces a method of probabilistic feature extraction that maintains a pipeline model. The authors user a measure of confidence on each node's output to select relevant features, and to analyze downstream feature dependencies.

A method from Raman et al., 2013 (12) builds upon probabilistic graphical model methods, generalizing them for data science tasks and attempting to improve the complexity of selecting to an optimal pipeline through several inference methods. The authors propose this as an alternative to hand-tuning every pipeline to particular task, instead constructing the pipeline as a DAG and following several paths, selecting the best under a criterion.

Interdisciplinary research & industry. With a greater means to collect data in novel ways, researchers are rapidly recognizing the need pipeline analysis and more powerful data analysis tools. Norgaard et al., 2018 (9) state this explicitly in their analysis of positron emission tomography data, used to capture the distribution of neurotransmitter activity. The authors find that their analysis leads them to question whether neuroimaging results can be determined invariant of prepossessing pipeline techniques.

Data-mindedness has always influenced industry as process optimizations are often sought for financial benefit. The proliferation of integrated IoT sensors has only heightened this interest; a paper form Oleghe et al. (2020) (10) demonstrates a framework for analyzing data collected throughout the manufacturing process using a pipeline, with the intention of formalizing the process as a whole.

6 Future Work

The implementation we present in this article is a proof of concept of the ideas behind it; it is a prototype. We can extend TTrees in several directions: The TTree library uses several machine learning libraries, it does not implement any Machine Learning, pre-processing, or visualization algorithms.

In the same way that TTree imports from several libraries, it exports pipelines to be used with other libraries. The most common libraries that implement pipelines are Scikit-Learn (1) and TensorFlow (2); exporting pipelines in the format required by those libraries is the first extension we devise.

A more general implementation that includes processing vector time series is also in our plans (10). Most application domains include related variables that could be used to forecast the variable of interest (based on other variables)

or forecast all variables at once. For example, in renewable energy, variables of interest include ambient temperature, humidity, atmospheric pressure, wind speed, wind direction, and solar irradiance. In several processes in that domain, we are interested in forecasting wind speed or solar irradiance; we could use the other variables to provide more information to produce such forecasts.

Executing a transformation tree be very time-consuming. Parallelizing the execution of those trees is a feature of interest to every data scientist. Nowadays, many libraries allow us to parallelize Python programs' execution (see (7; 1; 2).)

We also plan to provide a graphical user interface, where a click creates nodes, nodes are attached to other tree nodes by drag and drop, cutting old connections and establishing new ones. This kind of interface will allow the user to copy and paste sub-trees, select pipelines, and other operations related to TTrees. In general, the system will use several direct manipulation techniques to provide the user with a more natural experience designing transformation trees.

7 Conclusions

In finding a suitable pipeline to solve a data science problem, a data scientist searches for the best order and parameters of analysis, pre-processing, and modeling steps. This search process can be cumbersome and time-consuming; the search space grows with the number of transformation operations possible steps in the pipeline and the parameters those steps require. This article presents TTree (Transformation Tree), a Python library that allows a data scientist to deal with various options in producing a pipeline. Data science is a vast field; this article highlights the benefits of utilizing machine learning pipeline trees to process, analyze, model, and forecast time series data.

TTree forms a Transformation Tree that allows processing and manipulating the input however the data scientist sees fit. This library can simplify and expedite the process of time series analysis.

To the best of the authors' knowledge, this idea has not been presented in literature before. We expect to continue its development and integration with other machine learning libraries.

TTree has been converted to a package and uploaded to the pip repository for easy installation (install it using `pip install transformation-tree`). The source code has been been made publicly available on GitHub[3].

Acknowledgments

Juan Flores thanks the Department of Computer and Information Sciences of the University of Oregon for hosting him during his sabbatical leave. This work is a product of his stay at the CIS Dept.

[3] https://github.com/juanfie/time-series-modeling

Bibliography

[1] SKLearn Pipeline. `https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html`, accessed: 2021-02-25

[2] Tensorflow extended (tfx) is an end-to-end platform for deploying production ml pipelines. `https://www.tensorflow.org/tfx`, accessed: 2021-02-25

[3] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). pp. 265–283 (2016)

[4] Anderson, D.J.: Kanban: successful evolutionary change for your technology business. Blue Hole Press (2010)

[5] Bunescu, R.: Learning with probabilistic features for improved pipeline models. In: Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing. pp. 670–679 (2008)

[6] Cayley, A.: On the theory of the analytical forms called trees. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **13**(85), 172–176 (1857)

[7] Cingolani, P., Sladek, R., Blanchette, M.: Bigdatascript: a scripting language for data pipelines. Bioinformatics **31**(1), 10–16 (2015)

[8] Montgomery, D.C., Jennings, C.L., Kulahci, M.: Introduction to time series analysis and forecasting. John Wiley & Sons (2015)

[9] Norgaard, M., Greve, D.N., Svarer, C., Strother, S.C., Knudsen, G.M., Ganz, M.: The impact of preprocessing pipeline choice in univariate and multivariate analyses of pet data. In: 2018 International Workshop on Pattern Recognition in Neuroimaging (PRNI). pp. 1–4. IEEE (2018)

[10] Oleghe, O., Salonitis, K.: A framework for designing data pipelines for manufacturing systems. Procedia CIRP **93**, 724–729 (2020)

[11] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

[12] Raman, K., Swaminathan, A., Gehrke, J., Joachims, T.: Beyond myopic inference in big data pipelines. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 86–94 (2013)

[13] Saltz, J., Suthrland, A.: Ski: An agile framework for data science. In: 2019 IEEE International Conference on Big Data (Big Data). pp. 3468–3476. IEEE (2019)

[14] Wellner, B., McCallum, A., Peng, F., Hay, M.: An integrated, conditional model of information extraction and coreference with applications to citation matching. arXiv preprint arXiv:1207.4157 (2012)

[15] Cortes, Baldwin and Sanchez, Roberto and Flores, Juan J.: Characterization of a polycrystalline photovoltaic cell using artificial neural networks. Solar Energy **196**(1), 157–167 (2020)