

**Toward Better Model Structure Representation for Machine Learning Assistance in Computational Notebooks**

Submitted [Month YEAR], in partial fulfilment of

the conditions for the award of the degree **[your degree].**

**20320978**

School of Computer Science

University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

**Signature \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Date \_\_\_\_\_/\_\_\_\_\_/\_\_\_\_\_**

1. Introduction

Computational Notebooks, such as Jupyter Notebook, have been widely used by data scientists as tools for multiple types of data-based analytic or machine learning tasks. These coding platforms are especially suitable for data-relevant tasks, for user can execute single code cells instead of whole project, which tends to be flexible and convenient for exploratively data analysis, helping to discover data patterns and evaluate machine learning model performance. Combining multiple medias (code, text, plot, etc.) together, computational notebooks also enable users to obtain more instant feedback or evaluations comparing to traditional code editors.

However, in the process of exploring machine learning problems, data analysts may still find two main pain points:

1). It is hard to understand model structures from other’s code.

2). It is difficult to tackle with repeated experiments with high similarity in modelling and tunning process.

For pain point 1, Since data analysts or researchers may not have specific knowledge of the coding mechanics of machine learning models, when analysing model behaviours, these people may be confused and unable to extract model structure from the code. Researchers have also found that the flexibility of computational notebooks can often lead to disordered writing style of machine learning models, making the code messy, harder to read and explain **(Head et al., 2019)**, pushing this point to an even more irritating level. For pain point 2, as demonstrated by researchers, even in computational notebooks, which take supporting iterative exploration as a core purpose, scientists still must go through same process manually to do even simple modelling tasks **(Chattopadhyay, 2020)**. Since in this process, newer code and its execution results will replace former ones, when tunning or evaluating model parameters, the user may find it difficult to retrieve previous experiments for summarizing patterns from model performance histories. As stated by (**Liu et al., 2023)**, in the process of explorative coding by data scientists, as experiments and decisions are rapidly developed, abandoned and replaced, user’s ability to continually make quick and correct changes will be undermined. This situation can fit well in the tunning procedures of machine learning, where continuous changes to current parameters should be made based on previous model performances.

Current solutions for data analysts to solve these two pain points are limited. For pain point 1, it may be necessary for the analysts to track the markdown descriptions in notebook contents to infer model structures. This brings up the challenge that as current machine learning models are gradually increasing in scale, it often takes more time and efforts for these users to find and track multiple markdown cells which are often located far away from each other, especially in exploratory tasks, where corresponding documentations in notebook contents also tend to be personal, exploratory and messy **(Rule et al., 2018),** whichin most cases means markdown cells scattered at discrete locations around notebook contents, hard for user to focus on their relations and integrated semantics. For pain point 2, it may be necessary for analysts to record their former experiments, including data, model parameters and performance, etc. When taking notes from former experiments, the analysts tend to use tools or approaches apart from the computational notebook itself, such as version control systems or directly copying and pasting **(Kery et al., 2019)**, this in some circumstances makes the information more difficult to manage, and harder to obtain a clear view along with the notebook contents. In addition, even if the user has successfully collected all markdown cells indicating model structure, from fragmented information in these individual cells (even documented in detail), the user may still feel difficult to integrate them and build a full picture of the model structure. For example, an analyst may somehow gather all markdown contents indicating detailed information of each different model components, but they may find it difficult to link these model components together to form a workflow of the entire model, especially when the model complexity increases.

These challenges motivate the need for a tool to aid with both machine learning model structure comprehension and former experiments recording for data analysts or machine learning model evaluators. Based on the pain points mentioned above, the objectives of this tool will be:

1). to provide an intuitive visualization of model structure.

2). to enable user to record past experiments at any time of the exploring process within the computational notebook.

Enlighted by this need, a Machine Learning Helper, with model structure visualizations and experiment history recording functionalities, is proposed in this article as a Jupyterlab extension. This extension can provide assistance for users to achieve both the two objectives, and therefore release corresponding pain points in exploring and experimenting process. To achieve objective 1, the extension enables a model structure view automatically extracted from the markdown documentations of the notebook in forms of both table-of-contents and graphics for each notebook file. In order to support extra flexibility, the extension also allows user to add more components in the table-of-contents view to extend original documentations. To achieve objective 2, the extension enables user to append notes under any of the model components shown on the table-of-contents view of specified notebook file, including past experiment results or user’s own understanding of specific model components, these records will be stored in user’s local storage space, and will be loaded under the same model component at next time user returns to this notebook file. Comparing to previous works in model structure visualization, or history retrieving, this Machine Learning Helper extension integrates these functionalities together with more intuitive graphics view of model structures, making it more convenient for users to both easily comprehend model structures and review past experiments during data exploring or model tunning in machine learning process, without seeking help outside the computational notebook.

2. Related Works

2.1. Exploratory Programming in Computational Notebooks

Data scientists encountered difficulties in performing data-based complex tasks through traditional programming approaches. As a typical type of data-driven problems, one unique difficulty in machine learning, as discovered by researchers, is that because users cannot fully trust the accuracy of the predictions of the machine learning system and cannot have a direct way to prove its correctness, iterations in code seem to be inevitable **(Hill et al., 2016)**. To support the need of trying out different hypothesis, which is difficult by following traditional programming patterns, the concept of exploratory programming was introduced. Exploratory programming tends to be a programming workflow of exploring different approaches to obtain insights from data, and extend current goals based on these insights **(Subramanian et al., 2019)**. As discussed by Kery et al., when iterated experimenting must appear in coding for achieving hard tasks, or in scenarios where user must explore different possibilities of the program, exploratory programming often plays an important role for its attributes: flexibility, discovery and innovation **(Kery et al., 2017)**. However, though exploratory programming, or exploratory data analysis in data science context, tends to be the solution, and has quickly become an important part of modern data science workflow **(Batch et al., 2017)**, data scientists still need a coding platform which enables data exploration, as traditional code IDEs may not fully support this programming method.

In response to this issue, computational notebooks have been welcomed by data scientists. Commonly used computational notebooks, such as Jupyter Notebook, often consist of multiple individual cells for code or markdown documentations, users can run these cells individually and the execution of code in one cell will not be affected by contents in another cell. Also, computational notebooks contain instant feedback for the code cell which user just run, including plot visualizations, which even makes process of exploration more intuitive. These attributes enable users to rapidly prototype and share exploration results, making the computational notebook a suitable and widely preferred environment for exploratory programming **(Lau et al., 2020)**. Moreover, by combining text documentations in markdown cells, code in code cells and executed outputs (in forms of plot, tables, etc.) in one file, computational notebooks naturally provide expressive and interactive assistance for exploratory data analysis, and with the support of literate programming such as Python, the user can also integrate the exploration process into production pipelines **(Li et al., 2023)**. By using computational notebooks, data scientists can perform exploratory data analysis with more ease and convenience, and the lightweight and shareability of these notebooks enables data scientists to exchange ideas or hypothesis in open communities such as GitHub or Kaggle. However, as pointed by **(Wang et al., 2022)**, even with computational notebooks, the exploration still tends to be hindered by the characteristic of exploratory data analysis: active, rapid and diverse explorations, where proper documentations or explanations are shown to be tedious, and tracking previous experiments seems to be a huge workload for data scientists. These difficulties can obviously compromise the efficiency of exploration ideas sharing, and data pattern discovering in exploration process.

In this work, the Jupyterlab extension project aims to release these difficulties for data scientists by providing them an overview or explanation of model structures in notebook contents and enable them to record information about their past experiments in the data exploring process.

2.2. Model Structure Explanation in Computational Notebooks

Under the context that experiment results and exploration process sharing is now common between data scientists using computational notebooks, it is important to improve the ability of a data scientist to understand the exploration of other data scientists. In machine learning, this means to assist the readers of notebook documents to understand model structures and workflow. One obvious approach is to read the markdown documentations left by model developers, as the author of the notebook usually has most complete view and understanding of the model or their findings when exploring data. Well documented notebooks can provide most significant help for readers, for they can present the author’s discoveries to the readers in direct and intuitive ways, increasing the efficiency of data exploration sharing.

However, potential problems still exist. As demonstrated by Wang et al., in notebooks for exploratory data analysis, as hypothesis and alternative solutions change frequently, documentations usually contain a broad range of topics and purposes **(Wang et al., 2021)**. In a case study surrounding documentations of notebooks in Kaggle, they found that except for describing the contents of adjacent code cells, named Process, which occupies 58.65% of total content of the notebook, 32.36% of the content were used for organizing layout, specifying headlines of each part for navigation, and 19.19% of the markdown cells were used for describing and explaining results of code execution, some markdown cells were also used for explaining reasons of specific decisions, while some cells were mainly used for building to-do lists or summaries, explaining certain concepts, or providing references or meta-data **(Wang et al., 2021)**. Given this number of potential topics of documentation, which may be separated in different markdown cell in different locations of the notebook, user experience will probably be compromised while scrolling all along the notebook to track these cells to find information they desire for, especially in modern machine learning tasks, if without some tools to help them.

To solve this problem, model structure visualizations tools were presented by many previous works. For example, Jupyter presented official Table of Contents functionality, which will automatically generate a table of contents for user while launching the notebook by taking all user’s headings in markdown cells. InsideInsights, a web-based data-driven reporting system, aiming to help data explorers to express their exploration process, can provide a hierarchical structured data-flow view on an interactive canvas **(Mathisen et al., 2019)**. Albireo, developed by Wenskovitch et al., can build a graphic cell-level overview of the notebook, resolving relationships and similarities between different cells, and provide variable level navigations **(Wenskovitch et al., 2019)**. ToonNote enables user to convert their cells in notebook to an interactable “comic view”, where selected notebook contents will be summarized in a form of data comic in the extension panel **(Kang et al., 2021)**. These works, with algorithmic or programming approaches, provide users with clear visualizations of notebook structure based on user’s cell structure. The Machine Learning Helper extension in this article absorbed these advantages as part of the main functionalities, the extension has an interactive list-view of model components for user to dynamically navigate to corresponding cells, as well as adding new components for own designs. Also, an edible graphic interface is presented to users in a separate tab page of extension panel, providing more intuitive overview of model structure and overall dataflow.

2.3. History Recording and Retrieving in Computational Notebooks

In spite of the difficulties surrounding documentations while exploring data patterns with computational notebooks in machine learning process, data scientists found themselves in a situation where new alternatives rapidly replacing old ones, and when they wanted to find what they used to think of or experiment with, they often had no idea because there were not enough effective approaches to retrieve their old experiment records. It is useful for data scientists to record sufficient details for interrelated steps happened to produce the results, which can assist data scientists to track their exploring process **(Sandve et al., 2013)**, and as mentioned by Abdul et al., data scientists, especially when applying machine learning models, should be able to explain how they achieve current steps in analysis and this can help to better control the decisions made by themselves or by the algorithm **(Abdul et al., 2018)**. By being able to retrieve their own experiment histories, data analysts can get better knowledge of how they came over to current state, and better avoid potential mistakes they have made before. Meanwhile, by having more understanding of patterns emerged from these records, analysts can also control their model parameters, tunning them to achieve improvements of performance.

Though supporting history recording and retrieving is difficult in computational notebooks since one characteristic of computational notebooks is fast code executing and replacing, there are many previous works indicating to solve this problem. At first came the version control systems helping users to backup their previous notebooks, one of the examples is Variolite, where user can create and manage different versions of code blocks by drawing “Variolite boxes” around the desired code clock. Besides, one similar example from outside of computational notebooks is Azurite, a timeline-based code versioning plugin of Eclipse to help perform code backtracking, which means to help user going back to an earlier version of the code contents by removing later inserted code or restoring removed code **(Yoon and Mayers, 2015)**. To complement some inabilities of version control, including completeness and intuitiveness of history information, Verdant, a system enabling user to find each past performed choices, was presented. This system can provide powerful abilities to forage past notebook actions, inspect detailed history for partial and retrieve full notebooks in past versions **(Kery et al., 2019)**. In addition, Diff in the Loop, developed by Wang et al., supports visualizations of code and data differences between snapshot of past data versions dynamically during the data exploring process, which added another layer of flexibility while monitoring and controlling data exploration procedure **(Wang, Epperson, et al., 2022)**. These tools, by tracking past notebook actions through version control or history model approaches, can clearly improve relevant user experience.

However, while applying these automatic history tracking tools, it is always better to leave some space for users to manually record former experiments as notes. This can help user to cope with several emergent conditions and leave other messages for explanation except pure history information. In the Machine Learning Helper extension, user can take notes under model components in the list view of extension panel at any time of the exploration process, these notes will be saved for next visit, and user can delete them as like. Taking advantage of the time flexibility, this approach can supplement the disadvantage of some history tracking systems, since these systems usually wait for a check point to record history actions **(Kery and Myers., 2018)**, which on some occasions cannot fully perform their functionalities.

3. Description of the Work

The project in this article aims to develop a machine learning assisting system in form of a Jupyterlab extension, which can support data analysts using computational notebooks to:

1). Have a comprehensive overview of notebook contents based on author’s markdown documentations, in both table-of-contents and graphic views.

2). Correctly navigate to corresponding cells from table-of-contents view.

3). Add extra component blocks in the table-of-contents view, to provide flexibility for the user’s own documentations or understandings.

4). Be able to take notes to record experiment histories, or other supplement information such as hyper-parameter values.

4. Methodology

4.1. User Requirements

4.2.

Qualitative / quantitative

(task, notebook -> **design questions: rankings, comments, usability=why. 1 - 5**, use extension, extension feedback)

(compare => use & not use)

(how user use extension to finish task)

(3 – 5: qualitative)