Engineering a machine learning pipeline for automating metadata extraction from longitudinal survey questionnaires

Suparna De[1], Harry Moss[2], Jon Johnson[3], Jenny Li[3], Haeron Pereira[4], Sanaz Jabbari[2]

Abstract

Data Documentation Initiative-Lifecycle (DDI-L) introduced a robust metadata model to support the capture of questionnaire content and flow, and encouraged through support for versioning and provenancing, objects such as BasedOn for the reuse of existing question items. However, the dearth of questionnaire banks including both question text and response domains has meant that an ecosystem to support the development of DDI ready Computer Assisted Interviewing (CAI) tools has been limited. Archives hold the information in PDFs associated with surveys but extracting that in an efficient manner into DDI-Lifecycle is a significant challenge.

While CLOSER Discovery has been championing the provision of high-quality questionnaire metadata in DDI-Lifecycle, this has primarily been done manually. More automated methods need to be explored to ensure scalable metadata annotation and uplift.

This paper presents initial results in engineering a machine learning (ML) pipeline to automate the extraction of questions from survey questionnaires as PDFs. Using CLOSER Discovery as a 'training and test dataset', a number of machine learning approaches have been explored to classify parsed text from questionnaires to be output as valid DDI items for inclusion in a DDI-L compliant repository.

The developed ML pipeline adopts a continuous build and integrate approach, with processes in place to keep track of various combinations of the structured DDI-L input metadata, ML models and model parameters against the defined evaluation metrics, thus enabling reproducibility and comparative analysis of the experiments. Tangible outputs include a map of the various metadata and model parameters with the corresponding evaluation metrics' values, which enable model tuning as well as transparent management of data and experiments.

Keywords

automated metadata extraction, longitudinal surveys, machine learning, model provenance, hyperparameter tuning, DDI Lifecycle

Introduction

DDI-Lifecycle (DDI-L) (DDI Alliance, 2014) introduced a robust metadata model to support the capture of questionnaire content and flow and encouraged through support for versioning and provenancing objects such as 'BasedOn' for the reuse of existing question items. However, the dearth of questionnaire banks including both question text and response domains has meant that an ecosystem to support the development of DDI ready Computer Assisted Interviewing (CAI) tools is limited. Archives hold the information in PDFs associated with surveys but extracting that in an efficient manner into DDI-Lifecycle is a significant challenge.

Survey specification and development tools for standards-compliant questionnaire development using DDI-L require scalable and effective methods to enable automation of CAI. With a range of social sciences and biomedical domains' longitudinal studies forming part of CLOSER Discovery (https://discovery.closer.ac.uk), it offers a rich collation of questionnaire construct definitions and measurement approaches employed over a period, as well as scope for cross-study research. Due to the increased volume of data, with more questionnaires getting added to CLOSER Discovery, the ease, efficiency, and robustness of metadata extraction of question items are crucial considerations in

 De, Suparna; Moss, Harry; Johnson, Jon; Li, Jenny; Pereira, Haeron and Jabbari, Sanaz (2022). Engineering a machine learning pipeline for automating metadata extraction from longitudinal survey questionnaires, IASSIST Quarterly 46(1), pp. 1-12. DOI: https://doi.org/10.29173/iq1023

questionnaire processing, and ultimately, scaling to provide a high-quality question bank hosting survey questions for reuse by studies and data collection agencies. Use of question banks would have both a utility for discovery and through the accurate reuse of questions between studies, encouraging study and analyses reproducibility.

CLOSER has been annotating metadata from the study questionnaires in DDI-L. However, much of this has been done manually or semi-manually, making the extraction of structured metadata for data management purposes burdensome. To move away from manual processing of questionnaires and enable efficiencies in the survey process, automated methods are needed for questionnaire item metadata extraction.

Computational methods such as machine learning (ML) techniques, especially, supervised learning algorithms are an intuitive candidate approach for automating the extraction of valid DDI items from the survey questionnaires in PDF format that form part of CLOSER Discovery. The existing processed and marked-up (in XML) questionnaires form the training and validation dataset for applying supervised ML models. The extraction of the questionnaire items can be modelled as a text classification problem, distinguishing the questions, responses and instructions etc. as specific categories. Thus, this paper adopts a supervised ML approach for automated questionnaire item extraction from PDF survey questionnaires for inclusion in a DDI-L compliant repository.

Given a set of inputs and labelled outputs, supervised ML algorithms are geared towards allowing a model to learn over time, adjusting to minimise the error through a loss function. This necessitates a continuous build and integrate approach, with the different combinations of input data, feature engineering methods, model parameters and their resultant outputs being attached to an ML pipeline. This also requires capturing the various combinations experimented with, and the corresponding outputs, as metadata attached to the experiments, to ensure reproducibility, comparative analysis, and provenance of the pipelines. However, tracking the data and model transformations manually is time consuming and error prone. Existing initiatives such as IBM's PROV-ML schema (Souza, Azevedo, et al., 2019), that incorporate ML model aspects into the W3C PROV-O Recommendation (W3C, 2013), is a promising development in this regard. The open-source ProvLake (Souza, Mattoso, et al., 2019) Python library enables collection of provenance data related to function calls, with input arguments and output values captured. It provides a data tracker API which can be integrated into the machine learning workflow source code.

Thus, this paper also showcases the integration of the abstraction of model parameters through pipelines (through Data Version Control (DVC) (Kuprieiev et al., 2021)) and automating the process of attaching metadata related to each model experiment (through ProvLake). The process is illustrated with an implementation of the Naïve Bayes ML model which is an instantiation of a probabilistic classification algorithm. Tracking of model parameters, input data and output metrics is implemented in the hyperparameter tuning of the Naïve Bayes model where the DVC pipeline execution outputs a set of evaluation matrices, each corresponding to an individual variation in the hyperparameters. The different evaluation metrics for each parameter variation are then captured in a structured format, as a ProvLake log file. Other metadata recorded are the runtime information such as start time and end times of each data transformation execution.

Longitudinal Survey Questionnaires Dataset

Dataset Source

The dataset source was CLOSER Discovery (https://discovery.closer.ac.uk). The content is generated by a collaboration between CLOSER and its partner studies (Johnson, 2021) and contains metadata in DDI Lifecycle 3.2 for the questionnaires, datasets, study and data collection level information and a set of topics which are assigned to each question and resultant variable.

Training dataset preparation

The training dataset was extracted using Python 3 (van Rossum & Drake, 2009) code (Li, 2021) from CLOSER Discovery utilising the Colectica Repository REST API (Colectica, 2021).

The dataset included the DDI-L 3.2 items:

- question item, question text (QuestionName, QuestionLiteral),
- response domain items (CodeDomain, TextDomain, NumericDomain, and DateTimeDomain)
- conditionals (IfConditional, LoopWhile)
- interviewer instructions (InstructionText)
- statements (Literal)
- associated URNs for all of the above

The URNs allow the tracking and subsequent analysis of predicted values from different models.

The extracted train and test dataset was output as a tab-separated-value (TSV) file, to serve as input into the machine learning program. The entire process is illustrated in Figure 1 below.

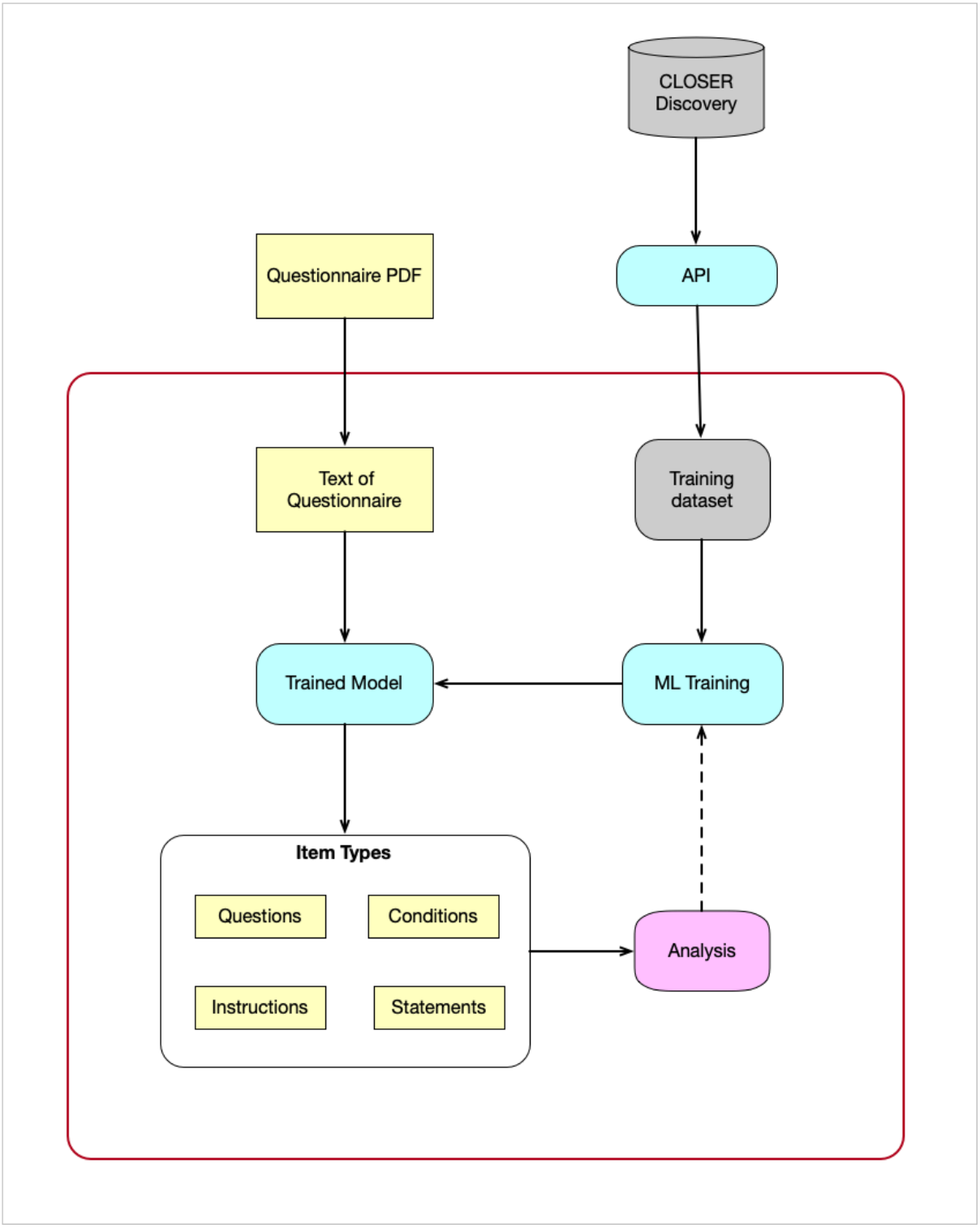

Figure 1: A schematic view of the training dataset generation from the CLOSER Discovery metadata store and model training.

Dataset Description

The dataset used as input into the machine learning models contains 187,105 rows, with the following item types (and their respective distributions), as shown in Table 1 below.

 De, Suparna; Moss, Harry; Johnson, Jon; Li, Jenny; Pereira, Haeron and Jabbari, Sanaz (2022). Engineering a machine learning pipeline for automating metadata extraction from longitudinal survey questionnaires, IASSIST Quarterly 46(1), pp. 1-12. DOI: https://doi.org/10.29173/iq1023

Table 1: Distribution of Items in Training Dataset

Item Type	Count
QuestionName	40,546
Question literal	40,545
Interviewer Instruction	3,051
Statement	7,551
Response Domain - Codelist	85,547
Response Domain - DateTime	486
Response Domain - Text	613
Conditional	8,414
Loop	352
Total	**187,105**

CLOSER ML pipeline

Git (Git, 2021) is used for version control of the underlying code used to pre-process input data, generate features for training from the data and for model training and evaluation. Additionally, text outputs of experiments and basic plots are versioned with Git. In this structure, each broad model family occupies a branch, with individual experiments represented by a directory containing output files following model training and evaluation.

The ML pipeline relies upon DVC to perform both dataset versioning and experiment tracking. In this context, an *experiment* refers to the model training process: from the choice of input parameters to the performance of the trained model on validation data according to several metrics of interest, such as accuracy, precision, recall, f1-score and the area under the receiver operating characteristic curve, referred to here as AUC score and ROC curve. Datasets versioned by DVC are referenced in the version-controlled codebase, managed by Git, and transferred via Secure Shell (SSH) to remote storage on the University College London (UCL) Research Data Storage Service (UCL, 2021).

Experimental Setup, ML Model and Code Versioning

DVC is used to version the state of the input data as it evolves and associate that with a specific git commit hash. DVC is also used within this work to transfer the versioned data over SSH to remote storage, from which it is accessible to other authorised users on the general-purpose high-performance computing (HPC) cluster at UCL. Model training is performed using a node on the UCL HPC cluster with a 36 core Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz and attached NVIDIA V100 GPU. A local environment with one AMD Ryzen 5 3600 CPU and one NVIDIA RTX 3070 GPU was additionally used for development purposes.

Datasets and trained models are stored, under DVC versioning, on the UCL Research Data Storage Service (RDSS). The RDSS is a petabyte-scale storage facility intended for research data storage in ongoing projects, includes data backup and is accessible via the HPC cluster described previously.

Dataset versioning is handled by DVC, which calculates a 32-character MD5 hash for each file within a directory and uses a JSON format file to record the relative locations of files within directories. The resulting hashes are stored in DVC configuration (.dvc) files, which are added to Git commits in order to pair input datasets to the relevant version of the codebase. An overview of the model, dataset and code versioning is provided below in Figure 2.

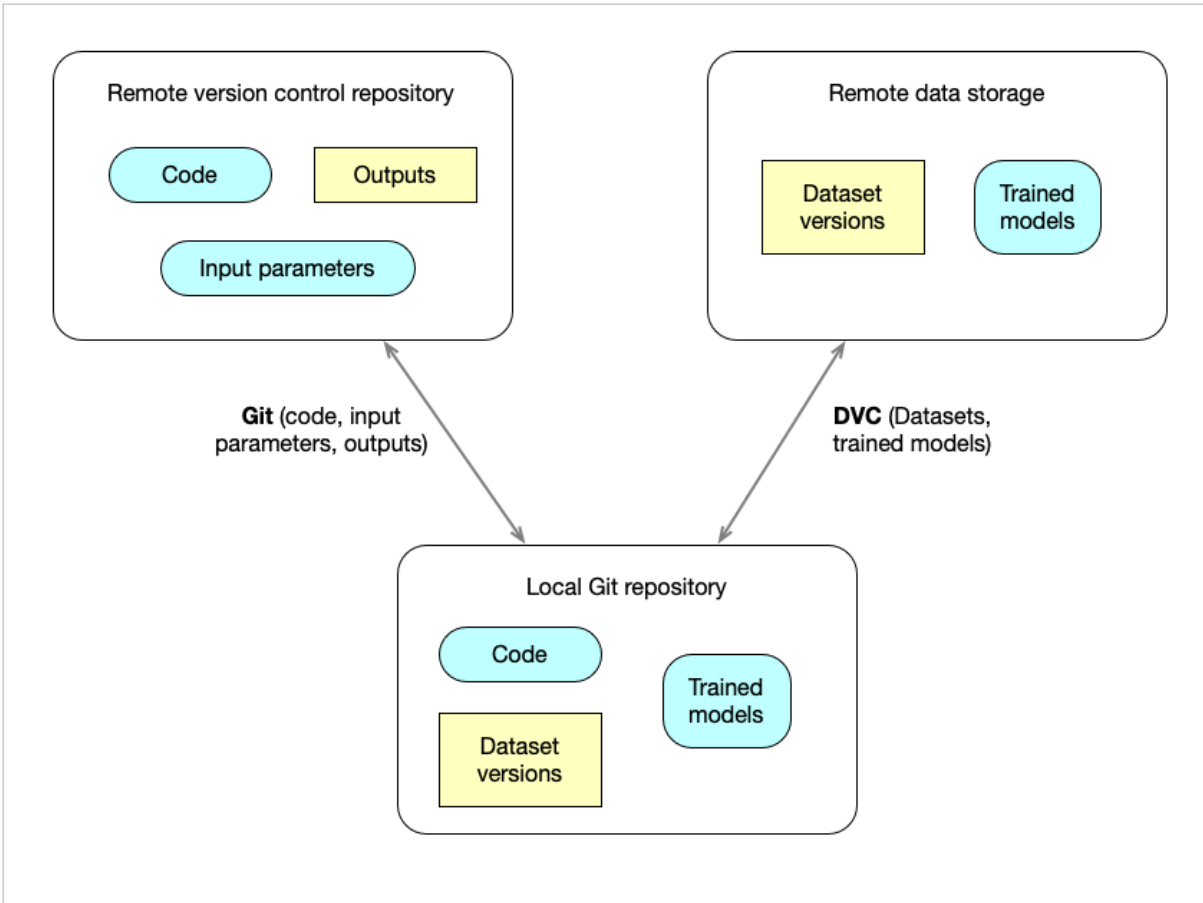

Figure 2: A schematic view of the relationship between Git and DVC version control and the local code repository. All code, documentation, input parameters, and DVC configuration files are version-controlled via Git and stored in a remote repository. Input datasets and trained models are versioned by DVC and are stored in a separate remote repository.

The codebase is entirely Python-based, relying on the PyTorch and scikit-learn machine learning libraries, and is broadly structured as follows, with some differences occurring between different model choices. In the top level of the repository, steering code exists to launch the relevant workflow via function calls and relies upon the correct setting of values in the parameters YAML (YAML Ain't Markup Language) file. The YAML file dictates the type of dataset to process, model type, output directory and file names and a range of hyperparameters for model training.

Hyperparameter tuning is concerned with choosing a set of optimal parameters which define the model architecture. In contrast with model parameters, hyperparameters cannot be directly trained from the data. The performance of a model can significantly vary according to the hyperparameters' values. Since finding the ideal hyperparameter can be time-consuming, search algorithms such as grid search and random search are typically employed. The key distinction between these two methods is the requirement to test all parameters. RandomSearchCV works with a few 'random' combinations (though typically with a set number of iterations) out of all the available combinations, whereas GridSearchCV scans all possible combinations. Both GridSearchCV and RandomSearchCV feature are functions available in Scikit-learn's model selection package and fit the model to the training set by looping through predefined hyperparameters. Both functions use the cross-validation method which divides the train data further into two parts - the train data and validation data to test the model for all possible combinations of the values given in the dictionary.

Once the correct workflow and initial arguments are provided, raw data consisting of text content and an item type category is read and stored as a pandas (Reback et al. 2021) DataFrame object. Input raw data is cleaned by removing underscores, renaming labels via regular expression matching and the removal of entries containing null entries for either text content or item type label. Following data cleaning, output directories for the experiment are created and the cleaned data is saved. Cleaned data is then passed as an argument to a function specific to each model type. At this stage, the model family is determined, and features are generated from the cleaned dataset in a specific manner for that model type. Data is split into 'train', 'validation' and 'test' sets, with a ratio defined in the input YAML. Text data is then tokenized and encoded, a process that converts individual words in the dataset into integer indices from a vocabulary used by the model. The data is further converted into a relevant format in the case of neural network-based models before commencing model training, using the model type defined in the input parameters YAML file. Following model training for a user-defined number of iterations, model performance is evaluated against the validation dataset. Output metrics and plots from model evaluation are saved to the output directory of the model along with a record of the input parameters used to obtain results.

Model runs are initiated via the DVC 'project' feature. Input parameters are provided by a YAML file (params.yaml) in a nested format, with the top level representing a 'stage' of the DVC project. By providing the DVC API with a project name, python file dependencies, expected output files, a file defining input parameters and a python script to run, DVC configuration files are produced following model training that records all of the above information and associates it with the project name, enabling reproducibility of results. In addition, when output metrics are tracked under DVC, running the same 'experiment' with differing parameters will display the effect on performance.

The results of model training and inference are stored either via Git on a remote GitHub repository or in the case of trained models, via DVC on the UCL RDSS. This project utilises the Git branch structure to separate different model families within the project, with separate output directories denoting different experiments. A pytest (Krekel et al. 2004) test suite is provided with the code to ensure changes to the code are error-free and can reproduce expected results after model training on a small test dataset. Changes to the code in the remote repository are performed using a Continuous Integration service via GitHub, which sets up a typical environment with which to run code tests and notifies the user if changes to the code will introduce breaking changes. In this way, it is also possible to check if changes to the code which at first may appear to be error-free in fact introduce changes to model behaviour.

Model Experiments

An example of the entire ML pipeline for the case of the Multinomial Naïve Bayes classifier follows, with an overview given below in Figure 3. Multinomial Naïve Bayes is a supervised learning

classification method that can be applied for categorical text data analysis. It is based on the assumption that each feature being classified is independent of all others and works by calculating the probability of each tag for any given text input, with the tag with highest probability forming the output.

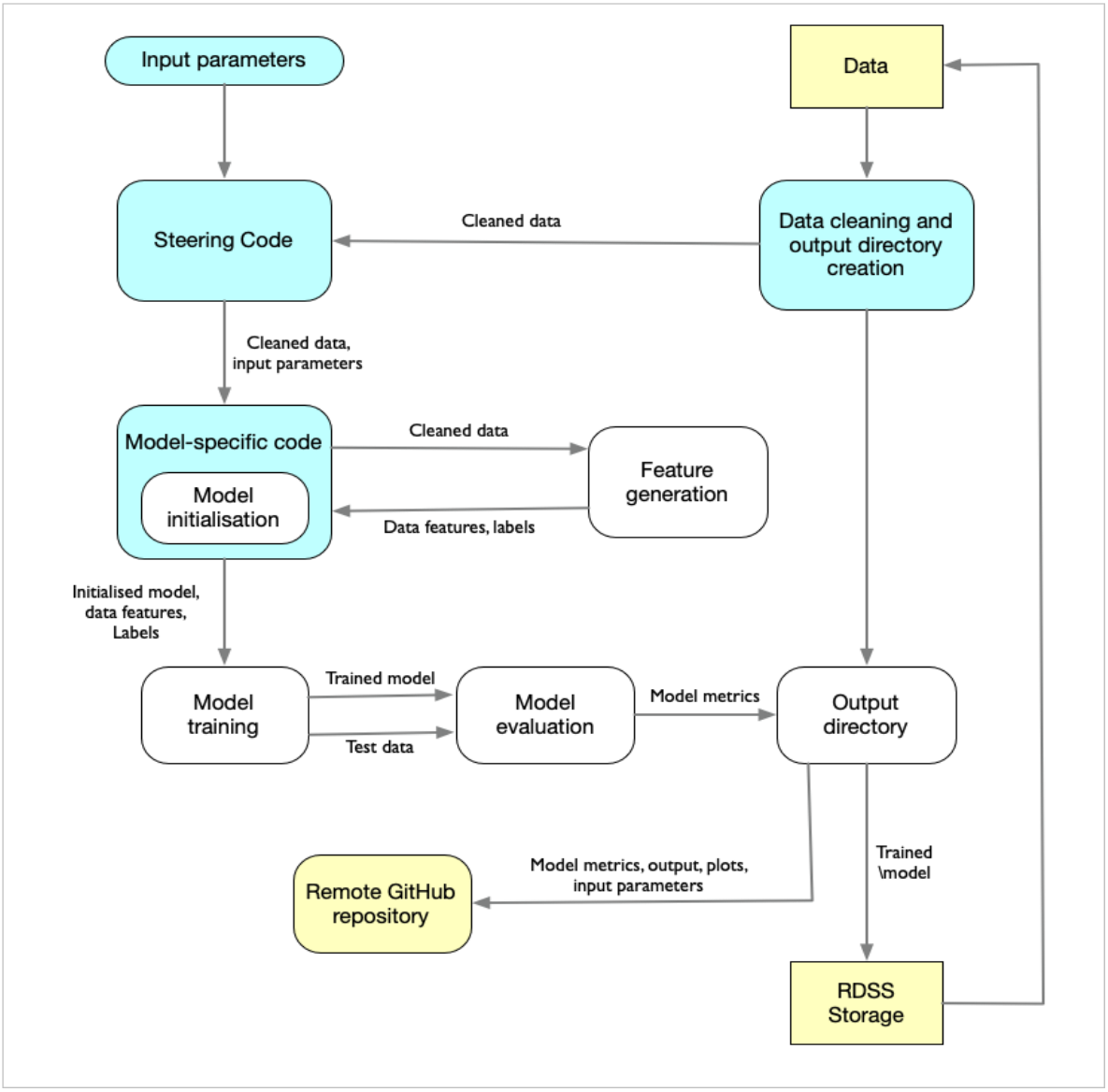

Figure 3: A schematic view of the stages of arbitrary model training from data ingest and cleaning to output of model results and their storage under version control via Git and DVC.

Input parameters, the dataset of choice and the 'MultinomialNaïveBayes' model are selected in the YAML file. Steering code is called which reads the parameters YAML file, performs data cleaning and sets up the directory structure to store model outputs. The cleaned data and parameters are passed as arguments to a specific Naïve Bayes classifier function, which handles feature generation, transformation of each text item into term frequency–inverse document frequency (TFIDF) vector representations and splits the data into training and test subsets. Multinomial Naïve Bayes classifiers are used in a one-vs-rest strategy, fitting one classifier per class. Performance is assessed during model training using a k-folds cross validation approach, where 'k' is the number of 'folds'. This approach defines a different subset of the training data as validation data in each 'fold' which is then used to assess the loss of the trained model. After validating model performance over all k-folds, the model is

then trained on the entire training dataset and its performance evaluated with the held-back test dataset.

Model metrics such as the accuracy, precision, recall, f1-score, AUC score, ROC curve plot and confusion matrix plot are saved to the output directory. Metrics scores are saved as nested JSON files, a per-class report and confusion matrix are saved as CSV files while ROC curve and confusion matrix plots are saved as SVG files. The trained model is exported as a Python joblib file and added to DVC version control, before manually being sent over SSH by the user to the RDSS. All other outputs are version controlled via Git and pushed to the remote GitHub repository.

Hyperparameter tuning is performed for the parameter *var_smoothing* which is a parameter added to the distribution's variance in the Gaussian Naïve Bayes model using GridSearchCV. Since the Naïve Bayes algorithm, with its Gaussian distribution assumption essentially gives more weights to the samples closer to the distribution mean, var_smoothing adds a user-defined value to the distribution's variance to account for more samples that are further away from the mean. The algorithm cross validates the model using each value for the parameter and outputs the best parameter combination with the help of the *best_params_* built-in function.

The accuracy, precision, recall and f1 score are calculated for each hyperparameter adjustment, and the ideal value is calculated. Provenance tracking is included in the hyperparameter tuning code, which keeps track of the parameter list and its associated evaluation metrics. This information is encoded in a ProvLake standard JSON format and is saved in a Prov ML log file. The ProvML log file is stored in the DVC directory with a standard naming pattern of 'prov' followed by the workflow name and the workflow execution start time.

The prov log file includes runtime details, input, and output data values in a standard format for every function in a workflow; additionally, we can keep separate prov log files for separate workflows, making analyzing data variation after each function execution much easier. For example, the log details of hyperparameter tuning function includes information such as unique id for that function, start time, generated time, end time, function name, the status of the run, Input parameters (list of hyperparameter value used), and output parameters (list of evaluation metrics for each hyperparameter variation). The structured format of the provenance log file is illustrated in Figure 4.

Figure 4: A snippet of the input and output data values as captured in a ProvLake JSON log file. Each row of the evaluation metrics' values map to the corresponding var_smoothing hyperparameter value.

The prov log file can be examined to gain a general understanding of the major functions used, the data variation they produce, as well as their particular runtime and fundamental run characteristics. In summary, these prov log files assist in keeping track of data and can be shared across team members to gain a thorough understanding of data flow.

Conclusions

Source code versioning, through tools such as Git, is well established in the software engineering community. But there are additional challenges in machine learning and data science, which require data version control as well as managing changes to models and datasets. Towards meeting this challenge, this paper has showcased a reproducible ML model training and execution method, which also generates logging metadata in a structured format, enabling tracking of various combinations of input data, model features and hyperparameter tuning with the obtained output values.

The developed ML pipeline is applied to automate the extraction of data and metadata from longitudinal survey questionnaires through a supervised machine learning pipeline approach. The pipeline approach employs Git for code versioning, DVC for model and data files versioning and links these proprietary methods in ML model versioning to an open provenance standard (ProvLake).

In addition to the aggregate measures, the rich provenance structure of the DDI-Lifecycle schema, allow the analysis of prediction of specific item types (e.g. question text) through the URNs linked to each input, which are inked to the specific questionnaire, study or if tagged to an ontology from which it originated, to give insights into where the predictions are more or less robust. These insights can be used to inform improvements in the models being used, and the potential need for more training data of specific types, the effects of different hyperparameter tuning approaches across both the whole training dataset and within specific subsets.

This further enables future comparative analyses, transparent data management, effective execution of experiments as well as provenance of the ML experiment settings.

Funding

Machine learning to enhance metadata in cohort studies. Science and Technology Facilities Council ST/S003916/1

Automating capturing structured content from questionnaires. ESRC ES/K000357/1

References

Colectica (2021). Colectica. Available at: https://www.colectica.com. Accessed 4 Oct. 2021.

DDI Alliance (2014). *DDI LifeCycle 3.2* [Computer software]. Available at https://ddialliance.org/Specification/DDI-Lifecycle/3.2/. Accessed 4 Oct. 2021.

Git (2021). git – fast-version-control. Available at: https://git-scm.com. Accessed 23 Nov. 2021.

Krekel, H. et al. (2004). *pytest 6.2.2*. Computer software

Kuprieiev, R. et al. (2021). *DVC: Data Version Control - Git for Data & Models (2.3.0)* DOI:10.5281/zenodo.4892897

Johnson, J. (2021). *Managing multiple UK longitudinal studies*. [online] Zenodo. Available at: https://doi.org/10.5281/zenodo.3775272. Accessed 4 October 2021.

Li, J. (2021). *Python interface to the Colectica API* (Version 1.0). Computer software. https://github.com/CLOSER-Cohorts/colectica_api. Accessed 4 October 2021.

Reback, J. et al. (2021). *pandas-dev/pandas: Pandas 1.2.1 (v1.2.1)*. Zenodo. DOI:10.5281/zenodo.4452601

Rogers, F.B. (1963). Medical subject headings. *Bulletin of the Medical Library Association*, [online] 51, pp.114–116. Available at: https://pubmed.ncbi.nlm.nih.gov/13982385/. Accessed 4 Oct. 2021.

UCL (2021). Research Data Storage Service (Research software repository). Available at https://www.ucl.ac.uk/isd/services/research-it/research-data-storage-service. Accessed 14 Oct. 2021.

Souza, R., Azevedo, L., Lourenço, V., Soares, E. F. d. S., Thiago, R., Brandão, R., Netto, M. A. S. (2019). *Provenance Data in the Machine Learning Lifecycle in Computational Science and*

Engineering, in WORKS 2019 - Workflows in Support of Large-Scale Science co-located with SC 2019 - ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis, Denver, USA.

Souza, R., Mattoso, M., Azevedo, L., Thiago, R., Soares, E. F. d. S., Santos, M. N. d., Valduriez, P. (2019). *Efficient Runtime Capture of Multiworkflow Data Using Provenance,* in eScience 2019 - 15th International eScience Conference, San Diego, United States.

Van Rossum, G. & Drake, F.L. (2009). Python 3 Reference Manual, Scotts Valley, CA: CreateSpace.

W3C (2013). PROV-O: The PROV Ontology. *W3C Recommendation*. [Online]. Available at: http://www.w3.org/TR/prov-o/. Accessed 14 Oct. 2021.

Endnotes

[1] Suparna De is a Lecturer in Computer Science at the University of Surrey and can be reached by email: s.de@surrey.ac.uk. (version: December 2021)
[2] Harry Moss and Sanaz Jabbari are with the Centre for Advanced Research Computing, UCL.
[3] Jon Johnson and Jenny Li are at CLOSER, UCL Social Research Institute.
[4] Haeron Pereira is in the Department of Computer Science at the University of Surrey.