Arangopipe, a Tool for Machine Learning Meta-Data Management

Jörg Schad [a], Rajiv Sambasivan [a] and Christopher Woodward [a]
[a] *ArangoDB*

Abstract. Experimenting with different models, documenting results and findings, and repeating these tasks are day-to-day activities for machine learning engineers and data scientists. There is a need to keep control of the machine-learning pipeline and its metadata. This allows users to iterate quickly through experiments and retrieve key findings and observations from historical activity. This is the need that Arangopipe serves. Arangopipe is an open-source tool that provides a data model that captures the essential components of any machine learning life cycle. Arangopipe provides an application programming interface that permits machine-learning engineers to record the details of the salient steps in building their machine learning models. The components of the data model and an overview of the application programming interface is provided. Illustrative examples of basic and advanced machine learning workflows are provided. Arangopipe is not only useful for users involved in developing machine learning models but also useful for users deploying and maintaining them.

1. Overview

The outlook for the adoption of machine learning-based solutions into technology-enabled aspects of business is strong[1]. This recent survey by McKinsey suggests that companies attribute their growth to the adoption of Artificial Intelligence(AI) in their application software. The companies that see the strongest growth are those associated with a core set of best practices and processes around developing and implementing AI-based solutions. Other surveys such as [2], [3] and [4] offer insights into the current state of affairs in developing enterprise machine learning solutions. These surveys suggest that reproducibility of results and tracing the lineage of activities performed on data science projects is a pain point in operationalizing machine learning solutions. This is consistent with the technical debt in the machine learning life-cycle discussed in [5]. One of the recommendations of [4] is the integration of open-source tools to circumvent these challenges. Reproducibility of results has plagued the research community too. [6] is recent work that exemplifies the brittleness of results and the need to promote rigorous methodology and tools to facilitate reproducible results. Therefore, tools that facilitate reproducibility can benefit both the practitioner and the research community. Not surprisingly, tools and methods to address this problem have attracted the attention of researchers in this area [7], [8], [9] and [10]. Arangopipe is a solution that was developed to address this pain point leveraging the unique features that a multi-model database supporting a graph data model offers in developing solutions for this problem.

Much of model development for machine learning and data analytic applications involves analyzing activities and findings that went into building earlier models, such as examining distribution characteristics of features, effective modeling choices, and results from hyper-parameter tuning experiments. Similarly, when these models are deployed, there is a frequent need to review data from previous deployments to verify configuration and deployment steps. These activities are those associated with reproducibility and traceability in surveys such as [4] and [2]. Therefore, applications and tools that record relevant information about machine learning model development tasks and facilitate the easy access and search of this information are of importance to teams involved in the development, deployment, and maintenance of machine learning applications. Arangopipe is a tool that provides these features. The process of developing and deploying machine learning models is often abstracted as a pipeline of activities. A graph is a natural data structure to represent the activities in the pipeline. Many machine learning tools and libraries routinely model the activities related to model development as a graph. There is great diversity in the range of machine learning applications and the complexity associated with developing these applications. Consequently, there is great diversity in data that needs to be captured from the development and deployment of these applications. A document-oriented data model is a good fit to accommodate this diverse range of data capture. With a document-oriented data model, there is no need to define the structure of the data before storing it. A database that permits both a graph and a document-oriented data model is ideal to capture data from machine learning model development and deployment activity. Arango DB provides these features. The rest of this article is structured as follows. In section 2, the nature of data collected from machine learning projects is discussed. In section 3, a series of illustrative examples illustrating the features of Arangopipe is provided. In section In section 4, work that is closely related to the problem solved by Arangopipe is discussed. The details of building and testing Arangopipe are discussed in section 5. In section 6, the salient observations and facts about using Arangopipe in data science projects are summarized.

2. Data Science Workflow

Arangopipe provides a data model that permits the capture of information about activities performed as part of a data science project. The information captured about the activity is also referred to as meta-data about the activity. The reference data model for Arangopipe was developed after critically reviewing the nature of meta-data captured across a range of projects from our own experience as well as standardization efforts such as [11] and [12]. While these efforts aim to standardize the operations and the data captured in productionalizing machine learning applications, it can be easily adapted thanks to the flexibility of the graph data model. A schematic representation of the data model is shown in Fig. 1. A graph data model is used to capture a comprehensive set and relations of activities that are performed in a data science project.

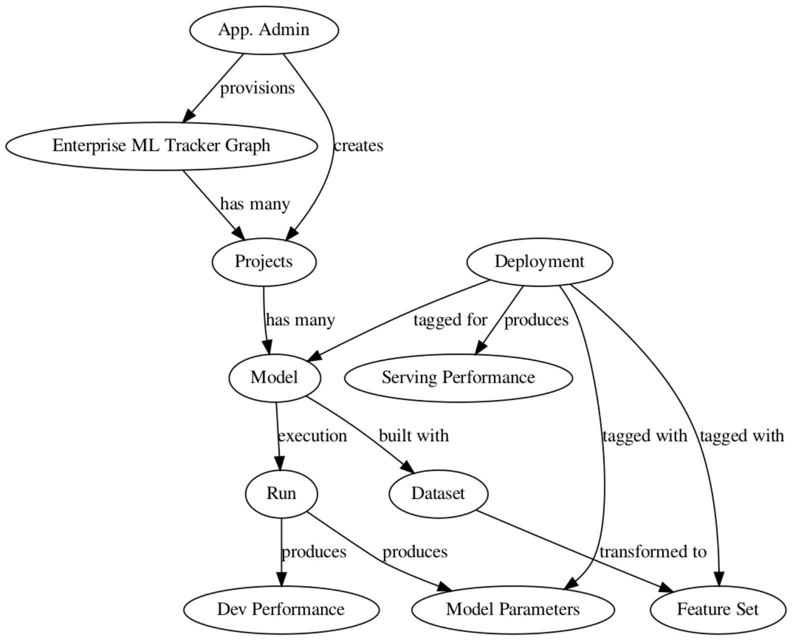


Fig. 1. Reference Arangopipe Schema

The purpose of each vertex type and its relevant relationships are as follows. Arangopipe provides two interfaces to capture and track data science activities. These are the administrative interface and the project interface. The administrative interface is called *arangopipe admin* and the project interface is called *arangopipe*. The administrative interface is used to provision arangopipe for use in an organization. The project interface is used to capture data from project activities. To use Arangopipe in your organization, the graph used to track machine learning and data science activities needs to be provisioned. This graph is called *Enterprise ML Tracker Graph* and is provisioned using the administrative interface. Machine learning and data science activities are organized by *projects*. The administrative interface is used to add machine learning projects that need to be tracked. Each *project* can track multiple model-building activities. A *model* is any data science activity that is tracked by the project. This can include a wide range of tasks performed by the data science team, such as:

(1) Data analysis experiments to profile attribute characteristics in a dataset
(2) Experiments to evaluate candidate models for a particular machine learning task.
(3) Hyper-parameter tuning experiments.
(4) Experiments to evaluate *data drift*.

Machine learning models use a *featureset* to train the model. A *featureset* is constructed from a *dataset* using transformations if required. A *run* captures the execution of a *model*. It links the inputs and the outputs associated with the execution of a *model*. A *run* executes a *model* with a set of *model parameters*. The model performance observed during the *run* is captured by the *dev performance* vertex type. A model with an acceptable level of performance is deployed to production. A *deployment* is used to capture such models. A *deployment* is created using the administrative interface. A *deployment* links the assets used by the deployed *model*. These include the *dataset*, the *featureset*, the *model,* and its *model parameters*. As a deployed model serves requests, we get to observe its performance. This performance is captured by the *serving performance* vertex type. See section 3 for the details of examples that illustrate how machine learning project activity can be captured with Arangopipe. It will be evident that each of the illustrative examples represents an instance of the data model described above. An example of an instance of this data model from a model-building experiment is shown in Figure 2.

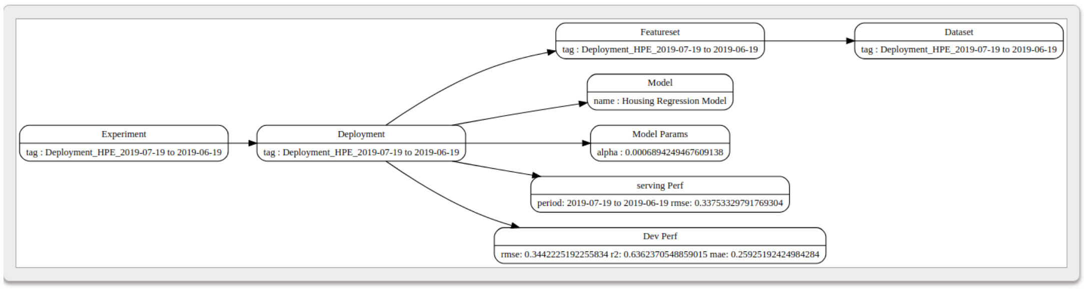

Experiment
tag : Deployment_HPE_2019-07-19 to 2019-06-19
Deployment
tag : Deployment_HPE_2019-07-19 to 2019-06-19
Featureset
tag : Deployment_HPE_2019-07-19 to 2019-06-19
Dataset
tag : Deployment_HPE_2019-07-19 to 2019-06-19
Model
name : Housing Regression Model
Model Params
alpha : 0.0006894249467609138
serving Perf
period: 2019-07-19 to 2019-06-19 rmse: 0.33753329791769304
Dev Perf
rmse: 0.3442225192255834 r2: 0.6362370548859015 mae: 0.25925192424984284

Fig. 2. Model Building Activity as a Graph

The same data model can be used to capture the meta-data from a variety of common machine learning experiments. Each of these examples follows the same template to capture meta-data. For new data assets, the properties of the asset are defined as a document. The asset is then registered and linked to the project. If it is an existing resource, it is retrieved using a search function and then its properties are updated. The data model discussed above should be sufficient for the meta-data capture from a wide range of experiments. However, if there is a need to introduce new elements into this data model for a specific application, that is also possible. For example, if you need to track a new type of meta-data for your machine learning experiments and you wish to capture this as a separate vertex type and create edges from existing vertices to the new vertex type, that can be done. Note that no schema constraint enforces the kind of data that you can capture with each vertex type described above. This benefit comes from using a document-oriented model for node data. This feature can be exploited for the following purposes. The same *model* vertex type can be used to capture models from multiple machine learning libraries. The *github* repository for this project contains examples that illustrate the use of identical workflow steps for model development tasks from different libraries such as *scikit-learn*, *tensorflow*, and *pytorch*. The artifacts and results produced from modeling tasks can be stored as node data. These results can be retrieved and re-materialized as programmatic objects. An example of this feature to store and re-materialize results from *Tensor Flow Data Validation* is available in the Arangopipe repository. In general, as long as the vertex data has a JSON representation and is not too large, for example, the weights associated with a massive neural network, it can be captured as part of the vertex data. To associate large data with vertices, the data can be stored in appropriate file formats such as *HDF5*, and the URL to the data file could be stored as a node property.

2.1. Software Implementation

Arangopipe consists of the following components:

2.1.1. `python-package` *interface*

The `python` package is primarily meant for data scientists to track and log data science activities they perform for projects. The package provides the administrative and project interfaces discussed above. A project administrator should first provision Arangopipe for use in the organization using the administrative interface. Provisioning sets up the database and graph to track machine learning projects as well as the connection to the provisioned database. The administrator then adds projects that need to be tracked to Arangopipe. At this point, project members can use the connection and project information to start logging model development activities.

2.1.2. Arango DB

This is the database used with Arangopipe. Project administrators and personnel involved in deploying and maintaining machine learning models can use the *Arango Query Language (AQL)* through the web interface of Arango DB to run queries on the Arangopipe database if needed.

2.1.3. A Web User Interface

A web interface to track, view, and search for information about assets and model building activity is provided. The web interface is primarily meant to be used by personnel deploying or managing machine applications to obtain historic information about model development activity, deployments, serving performance details, etc.

2.1.4. Container Images

Docker images of Arangoipe bundled with major deep learning toolkits, *pytorch* and *tensorflow* are available. These images contain the above components along with *pytorch* or *tensorflow*. Since these toolkits are widely used, data scientists and machine learning engineers can use these images to start using Arangopipe in their organizations.

It is possible to use Arangopipe with *Oasis*, Arango DB's managed service offering on the cloud. This would require no installations or downloads. The details of doing this are provided in section 3.

A schematic illustration of how the components discussed above are used with Arangopipe and ArangoDB is shown in Figure 3. An Arangopipe Administrator can use a Jupyter notebook to provision Arangoipe for a project. The administrator would use the *administration interface* for this purpose. Project team members, for example, data scientists, can use a Jupyter notebook (or a python script) to log meta-data about project activities using the *arangopipe interface.* This interface exposes two other interfaces. The *arangopipe_storage interface* offers functionality related to capturing meta-data from machine-learning pipeline activities in ArangoDB. The *arangopipe_analytics interface* offers functionality implemented with machine learning, such as methods to identify dataset drift. Dev-Ops personnel involved in operationalizing machine learning applications can use the web user interface to look up details of deployments or assets such as models or datasets. Arango DB offers an interface to develop analytic applications with ArangoDB. These applications can generate machine learning meta-data, for example, embeddings of graphs used in these applications. These applications can use the Arangopipe methods to store this meta-data in ArangoDB.

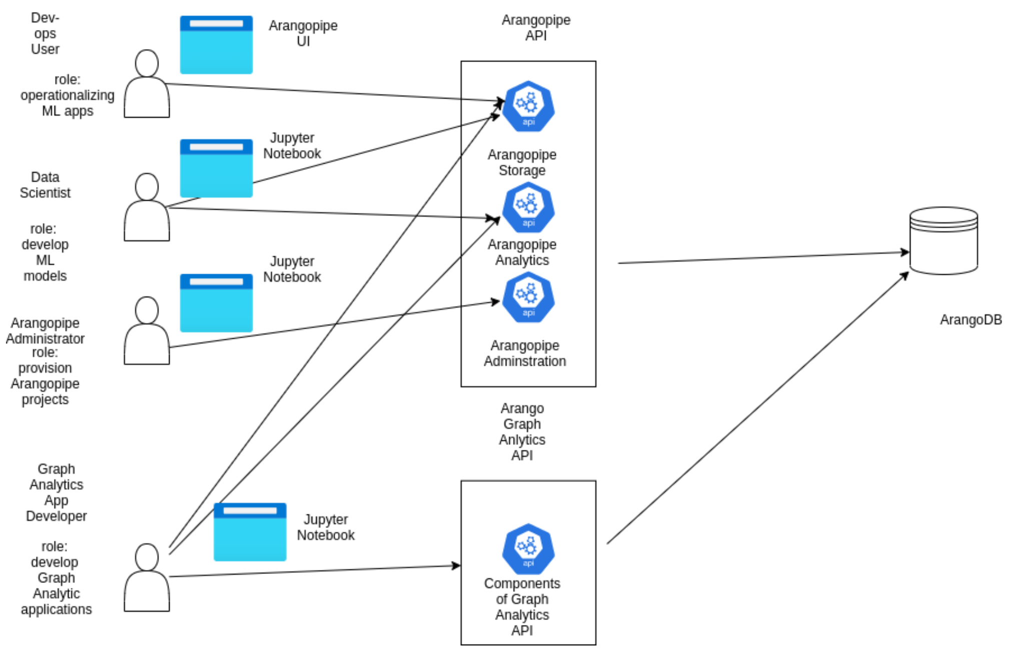

Dev-ops User
role: operationalizing ML apps
Arangopipe UI
Arangopipe API
Jupyter Notebook
Data Scientist
role: develop ML models
Arangopipe Storage
Arangopipe Analytics
Jupyter Notebook
Arangopipe Administrator role: provision Arangopipe projects
Arangopipe Adminstration
ArangoDB
Arango Graph Anlytics API
Graph Analytics App Developer
role: develop Graph Analytic applications
Jupyter Notebook
Components of Graph Analytics API

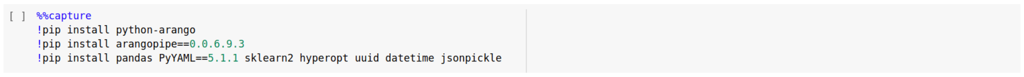
Fig. 3. Using Arangopipe with Analytics Applications

For more information about these components or to evaluate Arangopipe, please visit the project github repository [13]. The repository contains examples that illustrate the features of Arangopipe using Google's Colab [14] notebooks (see section 3). Working through these notebooks does not require the installation of software on your machine. These examples use Oasis, a managed service version of ArangoDB.

3. Illustrative Examples of Arangopipe

A comprehensive discussion of the advantages of using a graph data model to capture machine learning meta-data and a narrative detailing the progress of a data science project using Arangopipe is available in arangopipe-overview [15] and collaboration with arangopipe [16]. In this section, we present examples that illustrate the key features of Arangopipe. These features are illustrated with model building activity using the California Housing dataset [17] from the CMU statlib repository [18]. Each section below discusses a salient aspect of using Arangopipe with your model development efforts.

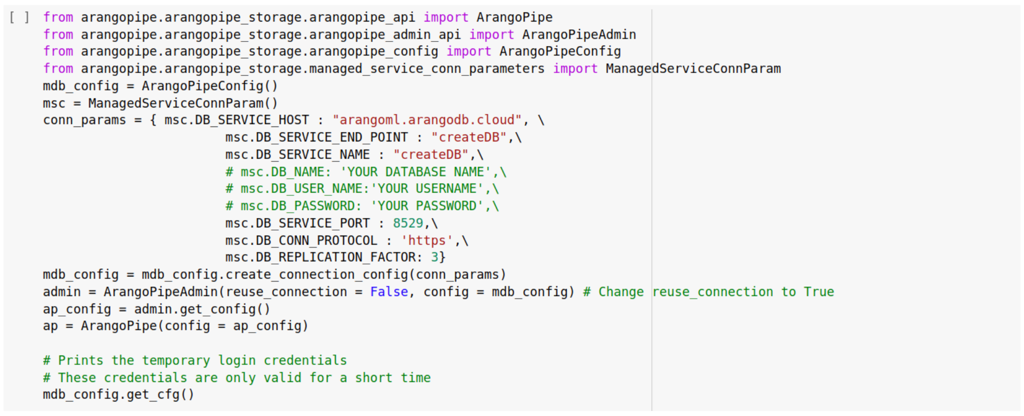
The basic template to use Arangopipe to capture machine learning meta-data from project activity is as follows. The first step involves installing Arangopipe and the dependencies needed for the machine learning project activity. An illustrative installation segment in a notebook is shown in Figure 4.



```
[ ] %%capture
    !pip install python-arango
    !pip install arangopipe==0.0.6.9.3
    !pip install pandas PyYAML==5.1.1 sklearn2 hyperopt uuid datetime jsonpickle
```

Fig. 4. Installing Arangopipe in a Jupyter Notebook

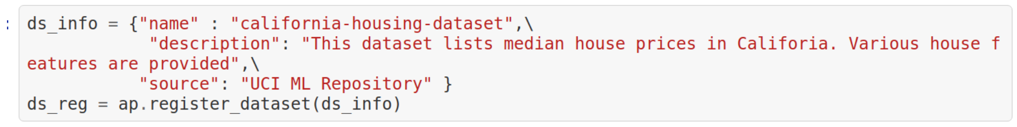
The second step is to provision Arangopipe for use with the project. This can be done with the administrative interface. An illustrative excerpt of code showing how this is done is shown in Figure 5



```
[ ] from arangopipe.arangopipe_storage.arangopipe_api import ArangoPipe
    from arangopipe.arangopipe_storage.arangopipe_admin_api import ArangoPipeAdmin
    from arangopipe.arangopipe_storage.arangopipe_config import ArangoPipeConfig
    from arangopipe.arangopipe_storage.managed_service_conn_parameters import ManagedServiceConnParam
    mdb_config = ArangoPipeConfig()
    msc = ManagedServiceConnParam()
    conn_params = { msc.DB_SERVICE_HOST : "arangoml.arangodb.cloud", \
                            msc.DB_SERVICE_END_POINT : "createDB",\
                            msc.DB_SERVICE_NAME : "createDB",\
                            # msc.DB_NAME: 'YOUR DATABASE NAME',\
                            # msc.DB_USER_NAME:'YOUR USERNAME',\
                            # msc.DB_PASSWORD: 'YOUR PASSWORD',\
                            msc.DB_SERVICE_PORT : 8529,\
                            msc.DB_CONN_PROTOCOL : 'https',\
                            msc.DB_REPLICATION_FACTOR: 3}
    mdb_config = mdb_config.create_connection_config(conn_params)
    admin = ArangoPipeAdmin(reuse_connection = False, config = mdb_config) # Change reuse_connection to True
    ap_config = admin.get_config()
    ap = ArangoPipe(config = ap_config)

    # Prints the temporary login credentials
    # These credentials are only valid for a short time
    mdb_config.get_cfg()
```

Fig. 5. Provision Arangopipe

After provisioning Arangopipe for a project, a project can register assets captured by the data model, for example, a dataset, using Arangopipe. An illustrative excerpt showing registering a dataset is shown in Figure 6.



```
: ds_info = {"name" : "california-housing-dataset",\
                "description": "This dataset lists median house prices in Califoria. Various house f
  eatures are provided",\
               "source": "UCI ML Repository" }
  ds_reg = ap.register_dataset(ds_info)
```

Fig. 6. Register a Dataset

Alternatively, a project can retrieve an existing asset, for example, a dataset using a lookup method, as shown in Figure 7 and then update its properties. In this scenario, a provisioned Arangopipe installation is used for the model tracking activity.

```
admin = ArangoPipeAdmin(reuse_connection=True)
the_config = admin.get_config()
ap = ArangoPipe(config=the_config)
# Read the wine-quality csv file (make sure you're running this from the root of MLflow!)
wine_path = os.path.join(os.path.dirname(os.path.abspath(__file__)),
                         "wine-quality.csv")
data = pd.read_csv(wine_path)

ds_reg = ap.lookup_dataset("wine dataset")
fs_reg = ap.lookup_featureset("wine_no_transformations")
```

Fig. 7. Lookup a Dataset

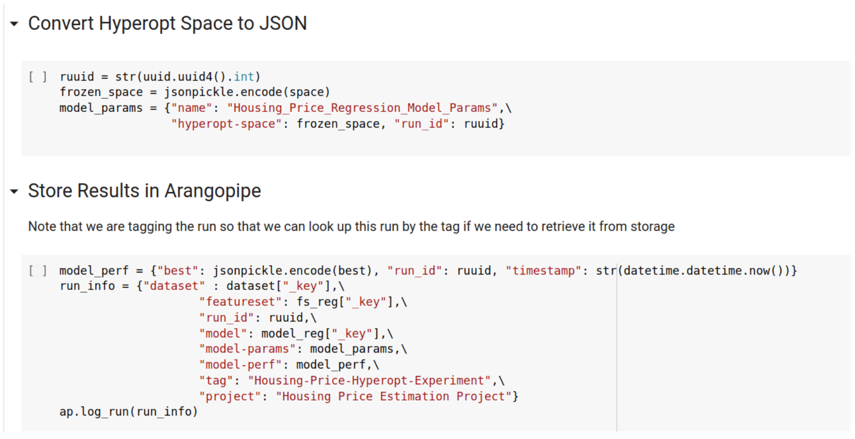
The examples that follow illustrate the use of Arangopipe in a set of representative application scenarios.

3.0.1. Basic Workflow

Please see Arangopipe Basic Workflow [19] for an illustration of the basic workflow with Arangopipe. It can be run as a *colab* notebook, so no installation is necessary to work through the notebook. The notebook develops a *Linear Regression* model using the *scikit-learn*[20] library and logs the results of the model development activity in Arangopipe. The notebook uses Arango DB's managed service offering, *Oasis*, as the database. The notebook execution begins with specifying a set of connection parameters to connect to the ArangoDB database instance to be used with Arangopipe. A method call is made through the administrative interface to provision the database for use with Arangopipe. This connection information can be saved and reused in subsequent interactions with the Arangopipe database instance. The project interface with which modeling activity is logged is then created with connection information used in the provisioning step. The notebook illustrates the typical administrative and project activities involved in using Arangopipe in a data science project.

3.0.2. Reusing Archived Steps

The sequence of steps illustrated with the basic workflow can be used to capture data science project activities from a variety of tasks. Data scientists can store key results from model building experiments in Arangopipe. These results can be programmatic artifacts, such as results from exploratory data analysis or a hyper-parameter tuning experiment. A colleague can retrieve these artifacts from Arangopipe in a subsequent session and re-materialize them as programmatic entities. Please see using Arangopipe with TFDV for exploratory data analysis [21] for an example of performing this in an exploratory data analysis task using *tensor flow data validation* *tensorflow data validation*. Please see performing hyper-parameter optimization with Arangopipe [22] for an example with a hyperparameter tuning experiment. The capability to store node data as documents is exploited in these examples. Node data and results from modeling are easily converted into JSON, which is the format that ArangoDB stores documents in. Tools to serialize programmatic objects into JSON and conversely, create programmatic objects from JSON, such as `jsonpickle`[23] are available. In some cases, like with `TFDV`, these are available from the tool used for the machine learning task. In a subsequent project activity, these model results can be retrieved from Arangopipe and re-materialized to programmatic objects native to the tool library that created them. Excerpts illustrating this for the hyper-parameter tuning experiment are shown in Figure 8 and Figure 9. The hyper-parameter tuning experiment stores the result of the experiment. The result can be subsequently retrieved.



Convert Hyperopt Space to JSON

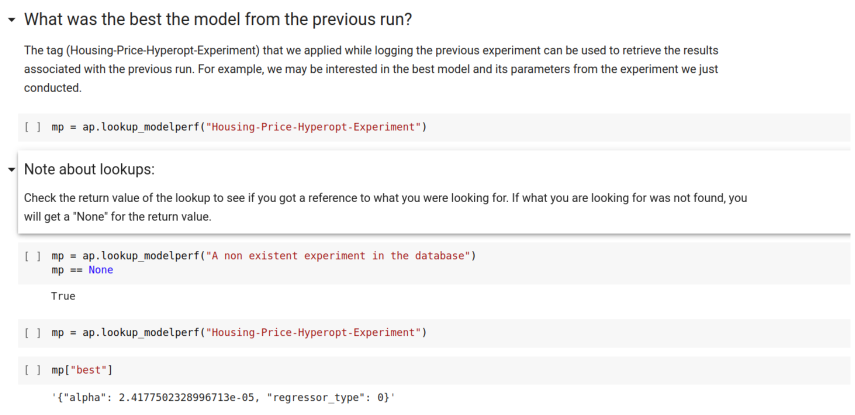
```
[ ]  ruuid = str(uuid.uuid4().int)
     frozen_space = jsonpickle.encode(space)
     model_params = {"name": "Housing_Price_Regression_Model_Params",\
                     "hyperopt-space": frozen_space, "run_id": ruuid}
```

Store Results in Arangopipe

Note that we are tagging the run so that we can look up this run by the tag if we need to retrieve it from storage

```
[ ]  model_perf = {"best": jsonpickle.encode(best), "run_id": ruuid, "timestamp": str(datetime.datetime.now())}
     run_info = {"dataset" : dataset["_key"],\
                      "featureset": fs_reg["_key"],\
                      "run_id": ruuid,\
                      "model": model_reg["_key"],\
                      "model-params": model_params,\
                      "model-perf": model_perf,\
                      "tag": "Housing-Price-Hyperopt-Experiment",\
                      "project": "Housing Price Estimation Project"}
     ap.log_run(run_info)
```

Fig. 8. Store Hyper-Parameter Tuning Result in Arangopipe



What was the best the model from the previous run?

The tag (Housing-Price-Hyperopt-Experiment) that we applied while logging the previous experiment can be used to retrieve the results associated with the previous run. For example, we may be interested in the best model and its parameters from the experiment we just conducted.

```
[ ]  mp = ap.lookup_modelperf("Housing-Price-Hyperopt-Experiment")
```

Note about lookups:

Check the return value of the lookup to see if you got a reference to what you were looking for. If what you are looking for was not found, you will get a "None" for the return value.

```
[ ]  mp = ap.lookup_modelperf("A non existent experiment in the database")
     mp == None

     True
```

```
[ ]  mp = ap.lookup_modelperf("Housing-Price-Hyperopt-Experiment")
```

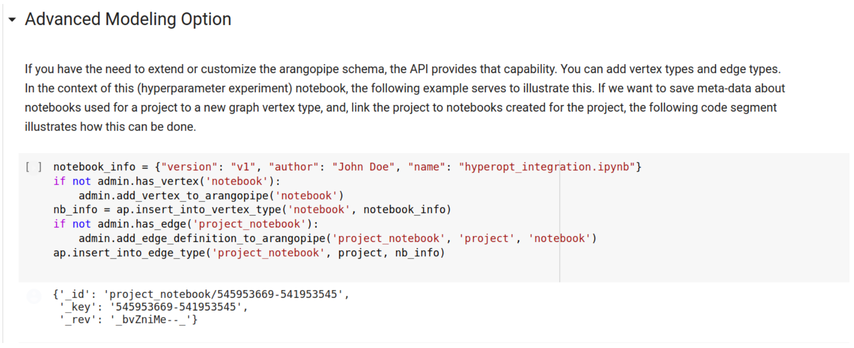
```
[ ]  mp["best"]

     '{"alpha": 2.4177502328996713e-05, "regressor_type": 0}'
```

Fig. 9. Subsequent Retrieval of Best Parameter Values

These examples illustrate another aspect of the utility of Arangopipe. Not all activities in a machine learning project are targeted at developing a machine learning model. There is much effort expended in activities like descriptive and exploratory analysis of the data, transforming the data to a form amenable for model building, experiments to determine parameters for model building, etc. Meta-data from these activities can also be captured by Arangopipe.

3.0.3. *Extending the data model*

It is possible to extend the Arangopipe data model to suit the custom needs that a project may have. For example, if an organization would like to capture notebooks as a separate asset that is tracked, this can be done. Please see the "Advanced Modeling" section of performing hyper-parameter optimization with Arangopipe for an example illustrating this. Projects that need to change or define a different data model in their projects can do so as illustrated in this example. This example illustrates the process of adding a vertex to the data model to capture project notebooks. An excerpt illustrating the code segment is shown in Figure 10.



Advanced Modeling Option

If you have the need to extend or customize the arangopipe schema, the API provides that capability. You can add vertex types and edge types. In the context of this (hyperparameter experiment) notebook, the following example serves to illustrate this. If we want to save meta-data about notebooks used for a project to a new graph vertex type, and, link the project to notebooks created for the project, the following code segment illustrates how this can be done.

```
[ ] notebook_info = {"version": "v1", "author": "John Doe", "name": "hyperopt_integration.ipynb"}
    if not admin.has_vertex('notebook'):
        admin.add_vertex_to_arangopipe('notebook')
    nb_info = ap.insert_into_vertex_type('notebook', notebook_info)
    if not admin.has_edge('project_notebook'):
        admin.add_edge_definition_to_arangopipe('project_notebook', 'project', 'notebook')
    ap.insert_into_edge_type('project_notebook', project, nb_info)
```

```
{'_id': 'project_notebook/545953669-541953545',
 '_key': '545953669-541953545',
 '_rev': '_bvZniMe--_'}
```

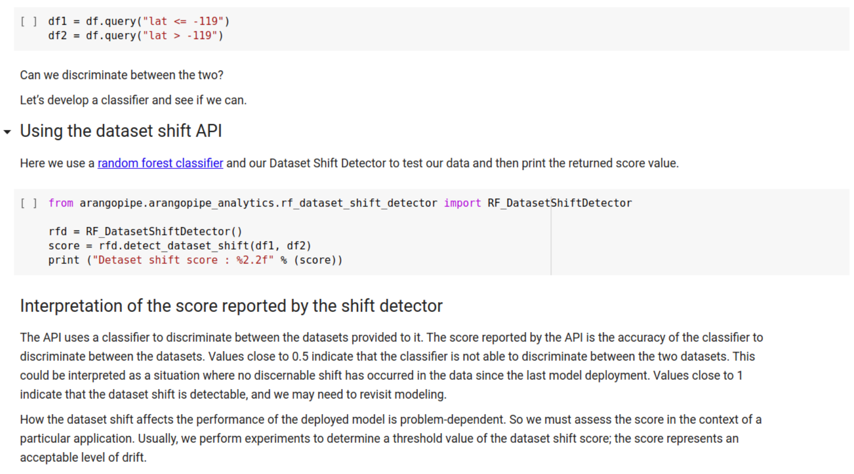
Fig. 10. Adding a Notebook Vertex to the Data Model

3.0.4. *Experimenting and documenting facts about models and data*

Documenting facts about models is a routine task for data scientists. The bias and variance of a developed model are of interest to data scientists on regression tasks. Please see using arangopipe to document model bias [24] for an example showing how model bias can be captured and stored with Arangopipe.

3.0.5. *Checking the validity and effectiveness of machine learning models after deployment*

Data drift and concept drift are known issues with maintaining and managing machine learning solutions. Arangopipe provides an extensible method to check for dataset drift. To detect dataset drift, a machine learning classifier is used to discriminate between the dataset used to develop the model and the data the model is receiving. A reference implementation using a RandomForest [25] classifier is provided. By implementing the abstract class, users can use the same idea, but use a different classifier for the task. An excerpt illustrating the use of this feature is shown in Figure 11. Please see capturing dataset drift with Arangopipe [26] for an example of using Arangopipe to evaluate data drift.



```
[ ] df1 = df.query("lat <= -119")
    df2 = df.query("lat > -119")
```

Can we discriminate between the two?

Let's develop a classifier and see if we can.

Using the dataset shift API

Here we use a random forest classifier and our Dataset Shift Detector to test our data and then print the returned score value.

```
[ ] from arangopipe.arangopipe_analytics.rf_dataset_shift_detector import RF_DatasetShiftDetector

    rfd = RF_DatasetShiftDetector()
    score = rfd.detect_dataset_shift(df1, df2)
    print ("Detaset shift score : %2.2f" % (score))
```

Interpretation of the score reported by the shift detector

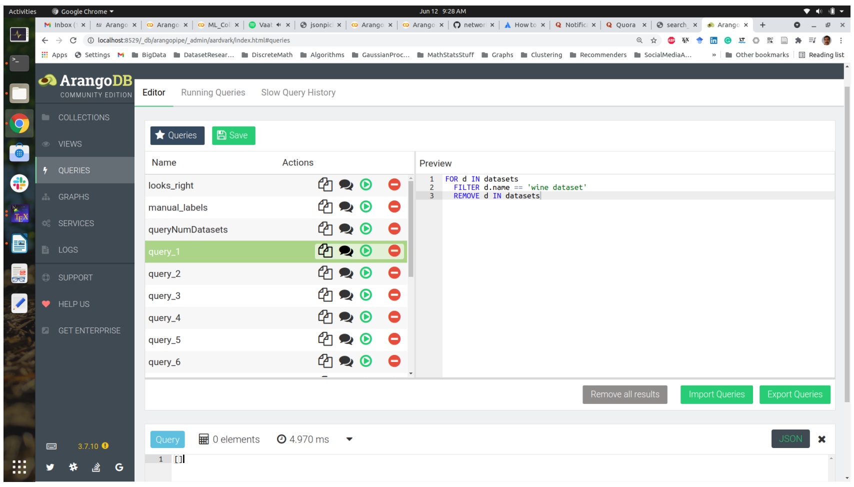
The API uses a classifier to discriminate between the datasets provided to it. The score reported by the API is the accuracy of the classifier to discriminate between the datasets. Values close to 0.5 indicate that the classifier is not able to discriminate between the two datasets. This could be interpreted as a situation where no discernable shift has occurred in the data since the last model deployment. Values close to 1 indicate that the dataset shift is detectable, and we may need to revisit modeling.

How the dataset shift affects the performance of the deployed model is problem-dependent. So we must assess the score in the context of a particular application. Usually, we perform experiments to determine a threshold value of the dataset shift score; the score represents an acceptable level of drift.

Fig. 11. Dataset Shift Detection with Arangopipe

3.0.6. *Using the Arangopipe Web User-Interface*

The easiest path to exploring the Arangopipe Web-User-Interface is by running one of the Arangopipe container images. Container images with pytorch[27] and tensorflow [28] are available. The project documentation [13] provides detailed instructions on launching one of these containers. After running the docker image, the web-user interface should be accessible. The section "Arangopipe User Interface" on the project documentation page provides information about the organization of the user interface and the details of using key features like *asset search* and *asset lineage tracking*. The Arangopipe Web-User Interface is primarily meant to be used by dev-ops personnel who operationalize machine learning applications. Searching for assets, tracing asset lineage, and obtaining information about past deployments are common tasks for personnel involved in operationalizing machine learning models. It is also possible to use AQL to query assets tracked by Arango DB. A comprehensive introduction along with detailed examples of using AQL to create and update data stored in ArangoDB is available at the AQL web page[29]. AQL queries may be executed from the Web-User-Interface. A detailed description of using AQL with the web user interface is available at AQL web interface documentation[30]. Figure 12 shows the web user interface that can be used to execute AQL queries.



```
FOR d IN datasets
  FILTER d.name == 'wine dataset'
  REMOVE d IN datasets
```

```
[]
```

Fig. 12. Web User Interface

3.0.7. *Storing Features From Model Development*

Feature engineering is very important in many machine learning tasks (Domingos, 2012). Arangopipe can be used to capture features generated from machine development. Graph embedding can be used to obtain a Euclidean representation of graph-structured data. These embeddings can be used as features in machine learning models. A variety of techniques exist to obtain these embeddings with each finding favor in particular applications [31]. Please see IMDB-Networkx-ArangoDB-Adapter[32] for an example of using Arangopipe along with the networkx adapter to store results from embeddings generated from node2vec [33].

3.0.8. *Support for R Models*

Using the *reticulate* R package, it is possible to capture meta-data from R data science tasks in Arangopipe. Please see an overview of reticulate [34] for information about this package and details of type conversion between R and python. Please see using R with Arangopipe [35] for an example of illustrating the capture of meta-data from R model development activity with Arangopipe.

4. Related Work

Efforts to standardize operations and data related to managing meta-data from machine learning were considered and are actively monitored in the development of Arangopipe. An example[36] illustrating the use of the elements of ML Spec in an Arangopipe model is available. The Open Neural Network Exchange [37] is a standard for the development of interoperable machine learning models. This standard defines a standard set of data elements and operations for a machine learning model. In this standard, as well as in most machine learning model development tools, the computation associated with developing the machine learning model is abstracted as a graph. Many tools to build data science pipelines such as Airflow [38] and Luigi [39] model the pipeline computation as a graph. In particular, the computation associated with the model or data science pipeline is expressed as a Directed Acyclic Graph. As execution flows through this graph, there is meta-data is produced. This is the meta-data that is captured by Arangopipe. Since a graph is used to express computation, a graph data model is a natural fit to capture meta-data about data science workflows. The meta-data are stored as documents in the nodes and edges of the graph. If there is a need to enforce schema constraints on the meta-data obtained as the computation progresses, this can be done. If the nature of meta-data obtained from the computation is dynamic and structural constraints for the data are not known apriori, the document-oriented storage model can offer the flexibility to capture such data. There is no need to define constraints about the data before it is captured as a document. In some data models, for example, the relational model, these constraints must be defined before the system starts ingesting data. This implies that structural constraints of the meta-data must be codified and set up before using the model to capture meta-data. The multi-model aspect of ArangoDB that permits the use of both a graph and a document-oriented data model to capture data is, therefore, a data model that provides both native expressiveness and flexibility for the capture of meta-data from machine learning pipelines. For the reasons discussed in section 1, the need to develop tools that facilitate reproducibility and tracking the provenance of models has attracted the attention of the research community. [40], [41],[8], and [42] are tools developed to solve the reproducibility problem. We believe that the graph and document-oriented storage models of ArangoDB offer considerable versatility and flexibility in capturing and analyzing meta-data from machine learning pipelines. As discussed, the provided data model was derived after reviewing solutions from our own experience, other solutions to the problem, a survey of research on this problem, and standardization efforts related to this area. If applications or projects need to use a different data model or extend the provided data model, Arangopipe provides methods for this purpose. As standardization efforts in this application area progress, new ideas that need to be captured by the data model can be added using the provided `python` package.

5. Building and Testing

The project repository [43] provides detailed information on the various options to try Arangopipe on a project. Arangopipe is open source, contributions are welcome! For details of building Arangopipe, please see the build instructions [44]in the project repository. Unit tests that illustrate how each method in Arangopipe is to be invoked are available [45]. For each method in the Arangopipe, a test case is available. A review of the test cases should provide the complete details of invoking each method in the Arangopipe. New users of Arangopipe can use the test cases as a reference for logging their project activity with Arangopipe. To facilitate exploration of Arangopipe and the Web UI, a test data generator [46] that provides example meta-data is also provided. The test data generation utility runs *linear regression* models on bootstrapped versions of the California Housing dataset and logs meta-data from model evaluations.

6. Conclusion

The capture of meta-data from the machine learning life cycle is important to both researchers and practitioners. Tools that capture and facilitate analysis of such meta-data are therefore useful to both these communities. The multi-model feature of ArangoDB presents some unique advantages in the capture and analysis of data from machine learning pipelines. A graph is a natural abstraction for this application since most tools used in developing machine learning pipelines model the computation as a graph. The document-oriented feature of ArangoDB offers flexibility in capturing meta-data from custom machine learning pipelines. Pipelines do not have to define the structural constraints about the types of elements in the meta-data obtained from machine learning pipelines before persisting it, as is the case for example with relational databases. However, if such constraints are desired, then it is possible to enforce them with a schema. The data model offered with Arangopipe captures the basic elements of any machine learning pipeline. This model is extensible and should a particular application need it, the package offers methods to make the desired changes. ArangoDB is tracking standards development initiatives around machine learning meta-data, such as ML spec. ArangoDB is committed to the further development of Arangopipe. Feedback and questions about Arangopipe are welcome in the ArangoML slack channel and the issues section of the project repository.

References

[1] Global survey: The state of AI in 2020 | McKinsey. https://www.mckinsey.com/business-functions/mckinsey-analytics/our-insights/global-survey-the-state-of-ai-in-2020.

[2] Algorithmia, State of Enterprise ML, a 2020 survey. https://info.algorithmia.com/hubfs/2019/Whitepapers/The-State-of-Enterprise-ML-2020/Algorithmia_2020_State_of_Enterprise_ML.pdf.

[3] Kaggle, State of Data Science and Machine Learning 2020. https://www.kaggle.com/kaggle-survey-2020.

[4] Anaconda, State of Data Science 2020. https://www.anaconda.com/state-of-data-science-2020.

[5] E. Breck, S. Cai, E. Nielsen, M. Salib and D. Sculley, The ml test score: A rubric for ml production readiness and technical debt reduction, in: *2017 IEEE International Conference on Big Data (Big Data)*, IEEE, 2017, pp. 1123–1132.

[6] X. Bouthillier, C. Laurent and P. Vincent, Unreproducible research is reproducible, in: *International Conference on Machine Learning*, PMLR, 2019, pp. 725–734.

[7] S. Schelter, F. Bießmann, T. Januschowski, D. Salinas, S. Seufert and G. Szarvas, On Challenges in Machine Learning Model Management, *IEEE Data Eng. Bull.* **41**(4) (2018), 5–15. http://sites.computer.org/debull/A18dec/p5.pdf.

[8] M. Vartak and S. Madden, MODELDB: Opportunities and Challenges in Managing Machine Learning Models, *IEEE Data Eng. Bull.* **41**(4) (2018), 16–25. http://sites.computer.org/debull/A18dec/p16.pdf.

[9] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S.A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie and C. Zumar, Accelerating the Machine Learning Lifecycle with MLflow, *IEEE Data Eng. Bull.* **41**(4) (2018), 39–45. http://sites.computer.org/debull/A18dec/p39.pdf.

[10] K. Katsiapis and K. Haas, Towards ML Engineering with TensorFlow Extended (TFX), in: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 3182–3182.

[11] S. Schelter, J.-H. Boese, J. Kirschnick, T. Klein and S. Seufert, Automatically tracking metadata and provenance of machine learning experiments, in: *Machine Learning Systems Workshop at NIPS*, 2017, pp. 27–29.

[12] ML Spec. https://github.com/mlspec/MLSpec.

[13] Arangopipe Project Repository. https://github.com/arangoml/arangopipe/blob/master/documentation/README.md.

[14] Google Colab. https://colab.research.google.com/notebooks/.

[15] Arangopipe, an overview. https://www.arangodb.com/2021/01/arangoml-series-multi-model-collaboration/.

[16] Arangopipe, a collaboration scenario. https://www.arangodb.com/2021/01/arangoml-series-multi-model-collaboration/.

[17] R.K. Pace and R. Barry, Sparse spatial autoregressions, *Statistics & Probability Letters* **33**(3) (1997), 291–297, Publisher: Elsevier.

[18] CMU Statlib. http://lib.stat.cmu.edu/datasets/.

[19] Arangopipe Basic Workflow. https://github.com/arangoml/arangopipe/blob/master/examples/Arangopipe_Feature_Examples.ipynb.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg et al., Scikit-learn: Machine learning in Python, *the Journal of machine Learning research* **12** (2011), 2825–2830, Publisher: JMLR. org.

[21] Tensorflow Data Validation. https://colab.research.google.com/github/arangodb/interactive_tutorials/blob/master/notebooks/ML_Collab_Article/example_output/Arangopipe_Generate_TF_Visualization_output.ipynb.

[22] Tensorflow Data Validation. https://colab.research.google.com/github/arangodb/interactive_tutorials/blob/master/notebooks/ML_Collab_Article/example_output/ML_Collaboration_Hyperopt_Integration_output.ipynb.

[23] JSON pickle. https://jsonpickle.readthedocs.io/en/latest/.

[24] Arangopipe Model Bias. https://colab.research.google.com/github/arangoml/arangopipe/blob/master/examples/Arangopipe_Feature_Example_ext1.ipynb.

[25] L. Breiman, Random forests, *Machine learning* **45**(1) (2001), 5–32, Publisher: Springer.

[26] Dataset Drift. https://colab.research.google.com/github/arangoml/arangopipe/blob/master/examples/Arangopipe_Feature_ext2.ipynb.

[27] Pytorch. https://pytorch.org/.

[28] Tensorflow. https://www.tensorflow.org/.

[29] Arango Query Language. https://www.arangodb.com/docs/stable/aql/.

[30] AQL Web Interface. https://www.arangodb.com/docs/stable/aql/invocation-with-web-interface.html.

[31] P. Goyal and E. Ferrara, Graph embedding techniques, applications, and performance: A survey, *Knowledge-Based Systems* **151** (2018), 78–94, Publisher: Elsevier.

[32] ArangoDB Networkx Adapter. https://github.com/arangoml/networkx-adapter/blob/master/examples/IMDB_Networkx_Adapter.ipynb.

[33] A. Grover and J. Leskovec, node2vec: Scalable feature learning for networks, in: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[34] Calling R from Python. https://rstudio.github.io/reticulate/articles/calling_python.html.

[35] Using R with Arangopipe. https://github.com/arangoml/arangopipe/blob/master/examples/R_Example_Colab.ipynb.

[36] Arangopipe ML Spec Integration. https://github.com/arangoml/arangopipe/blob/master/examples/MLSpec_AP_Example.ipynb.

[37] ONNX. https://onnx.ai/.

[38] Airflow. https://airflow.apache.org/docs/apache-airflow/stable/concepts.html.

[39] Luigi. https://luigi.readthedocs.io/en/stable/execution_model.html.

[40] D. Baylor, E. Breck, H.-T. Cheng, N. Fiedel, C.Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc et al., Tfx: A tensorflow-based production-scale machine learning platform, in: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1387–1395.

[41] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S.A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe et al., Accelerating the Machine Learning Lifecycle with MLflow., *IEEE Data Eng. Bull.* **41**(4) (2018), 39–45.

[42] J. Tsay, T. Mummert, N. Bobroff, A. Braz, P. Westerink and M. Hirzel, Runway: machine learning model experiment management tool, in: *Conference on Systems and Machine Learning (SysML)*, 2018.

[43] Arangopipe Project Repository. https://github.com/arangoml/arangopipe/tree/master/documentation.

[44] Arangopipe Build Instructions. https://github.com/arangoml/arangopipe/tree/master/documentation.

[45] Arangopipe Test Cases. https://github.com/arangoml/arangopipe/blob/master/arangopipe/tests/CItests/arangopipe_testcases.py.

[46] Arangopipe Test Data Generator. https://github.com/arangoml/arangopipe/blob/master/arangopipe/tests/test_data_generator/generate_model_data.py.

[47] K. Katsiapis and K. Haas, Towards ML Engineering with TensorFlow Extended (TFX), in: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 3182–3182.

[48] G.C. Publio, D. Esteves, A. \Lawrynowicz, P. Panov, L. Soldatova, T. Soru, J. Vanschoren and H. Zafar, ML-schema: exposing the semantics of machine learning with schemas and ontologies, *arXiv preprint arXiv:1807.05351* (2018).

[49] S. Studer, T.B. Bui, C. Drescher, A. Hanuschkin, L. Winkler, S. Peters and K.-R. Müller, Towards CRISP-ML (Q): a machine learning process model with quality assurance methodology, *Machine Learning and Knowledge Extraction* **3**(2) (2021), 392–413, Publisher: Multidisciplinary Digital Publishing Institute.

[50] P. Domingos, A few useful things to know about machine learning, *Communications of the ACM* **55**(10) (2012), 78–87, Publisher: ACM New York, NY, USA.

[51] A. Hagberg, P. Swart and D. S Chult, Exploring network structure, dynamics, and function using NetworkX, Technical Report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[52] D. Zheng, M. Wang, Q. Gan, Z. Zhang and G. Karypis, Learning Graph Neural Networks with Deep Graph Library, in: *Companion Proceedings of the Web Conference 2020*, 2020, pp. 305–306.

[53] A. Hagberg, P. Swart and D. S Chult, Exploring network structure, dynamics, and function using NetworkX, Technical Report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[54] D. Zheng, M. Wang, Q. Gan, Z. Zhang and G. Karypis, Learning Graph Neural Networks with Deep Graph Library, in: *Companion Proceedings of the Web Conference 2020*, 2020, pp. 305–306.

[55] T.p.d. team, pandas-dev/pandas: Pandas, Zenodo, 2020. doi:10.5281/zenodo.3509134.