Data and Machine Architecture for the Data Science Lab

Workflow Development, Testing, and Production for Model Training, Evaluation, and Deployment

Rosaria Silipo	Rosaria.Silipo@knime.com
Marco A. Zimmer	maz@mbrane.ch

Table of Contents

Data and Machine Architecture for the Data Science Lab ***Workflow Development, Testing, and Production for Model Training, Evaluation, and Deployment*** 1
Table of Contents 2
Summary 3
Introduction 3
Software Development vs. Model Building 3
  Processes vs. Prototypes 3
  CRISP-DM: the data analysis cycle 4
  Workflow Development for Model Building 5
Data and Machine Architecture 6
  Environments 6
  Sizing the Machines 9
  Roles in a Data Science Lab 10
Conclusions 10
References 10

Summary

When starting to put together a new data science lab, very few things are clear: only that we need predictive models and that we need them to run on real data. Everything else, particularly how these goals can be reached, is usually unclear and needs to take shape over time.

In this whitepaper we offer some advice about the script/workflow development process, predictive model building, deployment, and user access rights. Most of this advice is in the form of basic best practice guidelines, derived from our experience on a number of already running data science projects.

Introduction

When building a data science lab, as with any form of computing, we need to keep the process of software creation (development and testing) separated from the process of running that same software on real life data (production). In a data science lab, such software should cover the whole life span of a predictive model: training, evaluation, and deployment. In addition, different users, with different user roles and permissions, should be responsible for different parts of the software development cycle and/or of the model building cycle.

In order to provide first level help, we share a few basic guidelines here about how to implement the different phases of software development and model building, how to bind them to different virtual or physical environments, and how to guarantee access to some users and deny it to others, for any organization size and data science requirements.

Following these guidelines as early as the pilot phase, allows for easier scaling of the Data Science Lab IT architecture later, if/once it moves to a more sophisticated productive set-up.

Software Development vs. Model Building

The life of a data scientist involves switching between programming and mathematical models. Her/his daily routine consists of training and applying data-driven models to defined prediction tasks and of creating scripts and workflows to train and apply said models, maybe even “in real time”.

Processes vs. Prototypes

In a data science lab a number of workflows/scripts will generally be run on a daily (and nightly) basis for the purpose of preparing historical data, creating reports, training and evaluating models, applying models to real life data, possibly gaining insights about the underlying system, and taking consequent actions.

Some of these workflows run recurrently, some only once in the data science lab lifetime: this is the difference between processes and prototypes.

Prototypes

If you want a result only once and never again, there's not much to it - you just do it: fire up KNIME Analytics Platform (www.knime.com), create your workflows, gain your insight, save your results: be happy.

Ad hoc reports, one time data selections for specific campaigns, and one time only statistics all belong to the prototype workflow/script category. While it is important for the data science lab to work with a tool flexible enough to allow for agile and yet reliable prototyping, these kinds of workflows are not the main focus of this whitepaper.

Recurring Processes

Workflows implemented for recurring processes are run regularly and make the results available to a wider audience than just the workflow committer and the workflow developer. Such workflows need to be developed and tested thoroughly and accurately. Only after they have passed an exhaustive suite of test cases, to make them abundantly secure and bug free, are they ready to be moved into production. In production they will run regularly and produce results for others to consume.

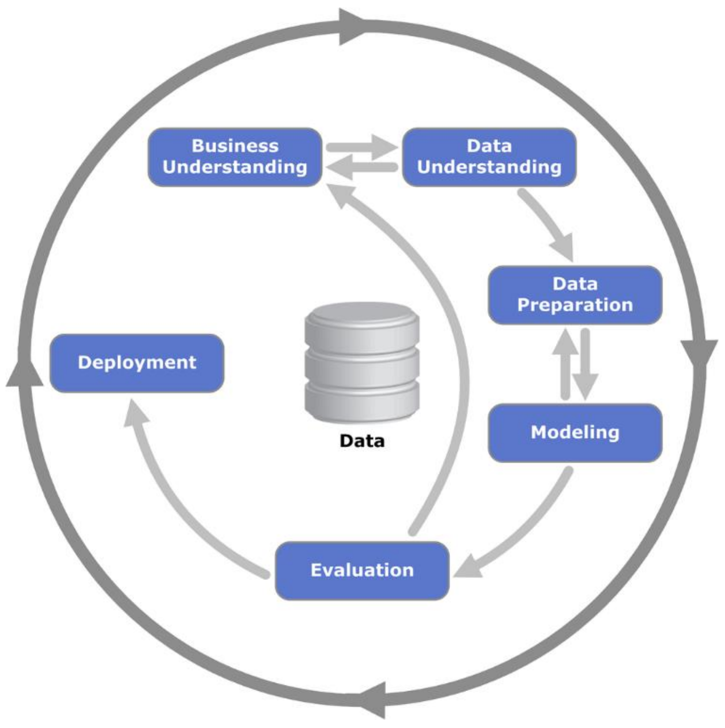
This whole "development - testing - production" journey needs to be monitored and must comply with a series of milestones, i.e. there needs to be a process in which all steps are clearly designed, milestones clearly identified, and resulting directions clearly defined.

CRISP-DM: the data analysis cycle

The majority of recurring workflows in a data science lab deals with creating and applying data-driven models, regularly and at fixed times. The creation of a data-driven model using data mining techniques is not a one stop shop only. It requires a number of trials to investigate the data, select the best model for the task, optimize the selected model, and so on. It requires a mixture of data mining knowledge, business knowledge, and experience in model optimization. The steps are not set in stone, but guidelines are supplied for example through the CRISP-DM process.

CRISP-DM [1] (CRoss Industry Standard Process for Data Mining) is a standard process for data mining solutions and it includes a number of stages: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment.

Figure 1. Phases of CRISP-DM by Wikipedia
(https://en.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining, CC BY-SA 3.0)



Assuming that we already have a clear business understanding of the project and, through a series of previous exploratory prototypes, the nature of the data is known well enough, we will focus here on the last 4 stages of the CRISP-DM life cycle.

Data Preparation includes all those cleaning, aggregation, and generally data manipulation operations to move from the original raw mess of data to a more structured and informative data table. A typical example in a CRM data set can be seen in the aggregation of values from a list of contracts into two groups of values, describing each customer in terms of revenues and loyalty.

Modeling includes the training of a model by means of a machine learning or statistical algorithm based on a set of data for a specific task, be it prediction or clustering. Modeling can be re-triggered after the first time, when the performance of the model becomes less satisfactory.

Evaluation requires testing the trained model performance on new data. This step is critical to get an idea of how the model will perform in real life conditions. Not only is this step necessary when creating the model the first time, but it needs to be regularly repeated to guarantee the model functioning properly over time. Data, like people, change over time. A successful model today might be obsolete in a few months. We need to re-evaluate its effectiveness periodically.

Deployment switches the model to run on (score) real life data at specific times. Sometimes the model might be required to score new incoming data in real time.

Workflow Development for Model Building

Each CRISP-DM phase requires a potentially different set of workflows to implement the task. We might need a workflow to prepare the data, a workflow to create and train the model, a workflow to test the model, and a workflow to deploy the model. Each one of these workflows needs to be developed, tested, and finally executed on real life data, like any other script or program. That is, each workflow implementing a CRISP-DM phase has to go through "development", "test", and "production".

Workflow / CRISP-DM	Development	Test	Production
Data Preparation	Create a workflow to load, clean, enrich, deal with missing values, and transform your data.	Test the workflow for errors against new and unexpected data situations, e.g. more columns.	Run regularly in a DWH production environment to collect, prepare, and structure all incoming data.
Model Training	Define task and extract a sufficiently representative training set. Create a workflow to train a model on the training set.	Verify that the workflow trains the model even when the data change, e.g. with more missing values.	Create new models, regularly or in the event that previous models become obsolete or not existent.
Model Evaluation	Create a workflow to assess and monitor model performance. Define lowest tolerable performance limit.	Verify that the workflow directs process to re-training or deployment correctly, even when the data change.	Assess model quality periodically.
Model Deployment	Create workflow to apply accepted model to "real life data".	Check workflow performance in real life conditions (speed, changing data, locale, etc...).	Process data regularly for real and send results for decision making.

In order to avoid confusing terms, in the following section we will distinguish between "model training", "model evaluation", and "model deployment" on the one hand and "workflow development", "workflow testing", and "workflow production" on the other. The table below shows the three different stages (development, testing, and production) required for each workflow implementing a phase in the CRISP-DM process.

It is good practice to separate training from deployment in two separate workflows. Deployment is run frequently: whenever new data comes in. Training, in comparison, is triggered only to run when models prove to be obsolete. Separating the two phases allows for a faster execution of the deployment workflow.

Data and Machine Architecture

The data and machine architecture must mirror the workflow development phases, which, as we have seen, are different from the model building phases. Workflow development phases include development, testing, and production.

It is good practice to keep the workflow development phases in separate environments. This, for example, prevents polluting production data with occasional development mistakes.

Environments

Environment separation means segregated data and workflow regions - or even machines. In addition to the three mentioned environments, we could optionally add a fourth one for pre-production / fail over / fall back. This last environment is usually designed to take over production if a catastrophic event hits the production region or machine.

The segregation can be done logically (separated regions on the same machine) or physically (separated machines); it must cover the workflows, data, and meta nodes, and needs dedicated environments for:

- Development
- Testing
- Production
- Pre-Production (PreProd)/Fail-over/Fall back (optional)

Machine and Data Architecture

Production

The production environment is the most delicate one, since it runs the workflows for real on real data. This means it has to be strongly protected from external intrusions and updated regularly.

In production, a full DataWareHousing (DWH) solution is usually necessary, requiring remote execution, scheduling, rollover, versioning, user authentication, access rights, resource sharing, interactive dashboard, and communication through a REST/API interface.

This is indeed the phase in which the enterprise features of KNIME Server are of particular benefit in order to guarantee a reliable orchestration, scheduling, and execution of the processes, the environment integrity, and resource sharing.

Leveraging the openness of the KNIME platform, it is also possible to mix and match execution engines within a KNIME workflow.

For example, the first pre-processing phase might run completely in-database on a big data platform using the nodes for **in-database processing**, i.e. database connectors and database manipulation nodes. Running the pre-processing part of the workflow on a big data parallelized platform could speed up the workflow execution.

It is also possible to run R, Python, Spark, Weka, and any other external tool or script from within a KNIME workflow. This feature extends KNIME functionality to include compatibility with all available file formats and machine learning algorithms.

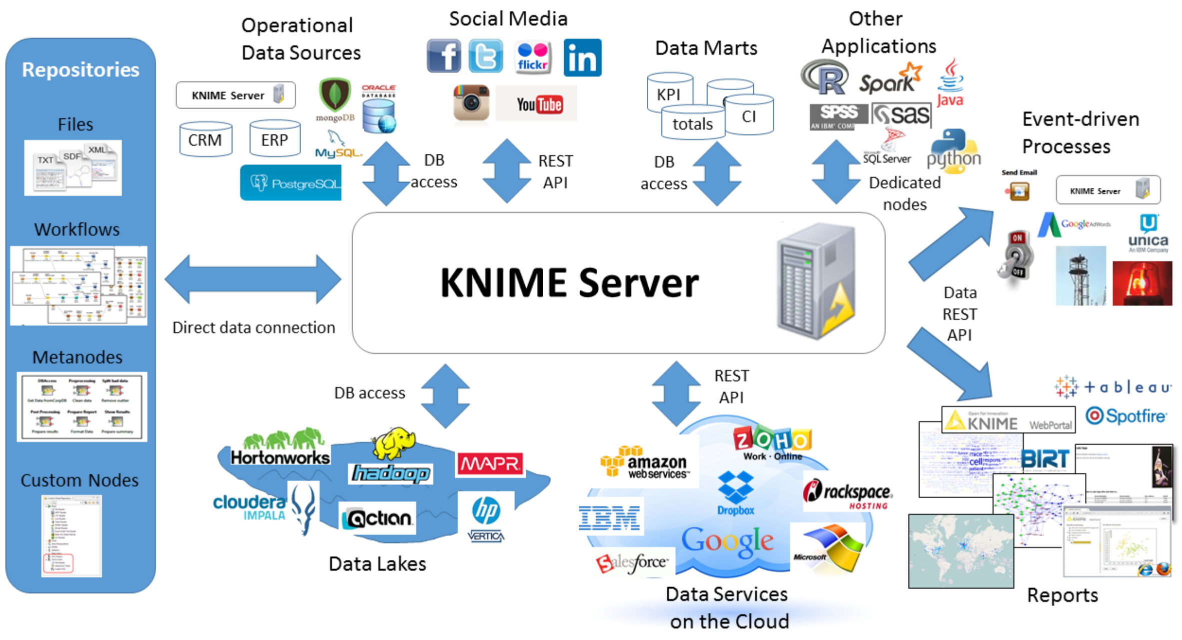
Given KNIME's abundant capabilities to connect to different sources and tools, it can also act as a "layer of indirection" to decouple your projects from the fluctuating technology stack.

The production environment needs to access different data sources: files and databases locally and in the cloud; data lakes; results from external applications via files or REST services; results from other KNIME workflows, again via files or REST services; operational data from CRM, ERP, and other similar storage tools; traditional data marts; web APIs; social media; remote repositories; and probably more by the time this whitepaper is published.

The production environment also needs to access and/or trigger a wide range of different tools: big data platforms; streaming engines; analytical tools; statistical tools; business knowledge; business related applications; REST services; API services; and more.

The figure below shows a possible instance of a production environment, with probably just a subset of the required connections to data and tool sources.

Figure 2. Data and Service Architecture in Production, centered on KNIME Server



Pre-Production/Fail Over / Fall Back

Pre-Production/Fail Over/Fall Back environment is usually an exact, separate copy of the production environment, updated as regularly in terms of workflows, data, and metanodes as the production environment.

Development

The same architecture as in production and in pre-production is replicated on a smaller scale for the development and testing environments.

The development phase needs a location where all of the temporary workflows currently under development can be stored. This location can be the open source KNIME Analytics Platform on the developer's machine.

However, KNIME Analytics Platform alone does not provide for the sufficient sharing of resources. There the only possible way to share workflows across different instances of KNIME Analytics Platforms is to use the *Import / Export Workflow* option in the *File* menu. Similarly, the only possible way to reuse meta-nodes is through copy and paste.

A more professional approach to workflow development requires a more flexible feature for resource sharing. This can be obtained via KNIME Team Space or KNIME Server, depending on the development group size.

A separate set of data has to be used for development. This does not need to be an exact reproduction of the productive data set. A data subset of sufficient generality, updated every now and then, would probably suffice.

Testing

The testing environment needs to run test cases workflows on test data, collect results, and produce a stability report.

Let us now talk briefly about workflow testing, even though most programmers are already familiar with it. Testing can assume many different forms.

- *Unit testing*: small test samples with defined outcome to make sure that the workflow works as it should in all its pieces
- *Regression testing*: test samples aimed to assess whether the workflow still works as in the past or whether we have introduced new unknown bugs
- *Stress testing*: test samples for speed and performances under stress conditions (overloaded machines or large amounts of data for example)
- *Automation*: scripts to run previous tests regularly and automatically, generating alarms when results do not compare as they should

About testing, in the KNIME Node Development Tools extension, part of the open source release, you can find a "**KNIME Testing Framework**" package, including a "Testing" category with a number of nodes specifically developed for workflow testing. This extension is part of the free KNIME Analytics Platform.

After installing the extension, the "Testing" category appears in the Node Repository. This kind of category contains nodes such as:

- Fail in Execution, to simulate failing nodes
- A number of Difference Checker nodes (Table, File, PMML, Model Content, Image, …), to compare with gold standards
- Test Data Generator, to create test data
- Table Validator, to fail the test if some pre-defined conditions are not met

- And many other nodes to help with workflow testing

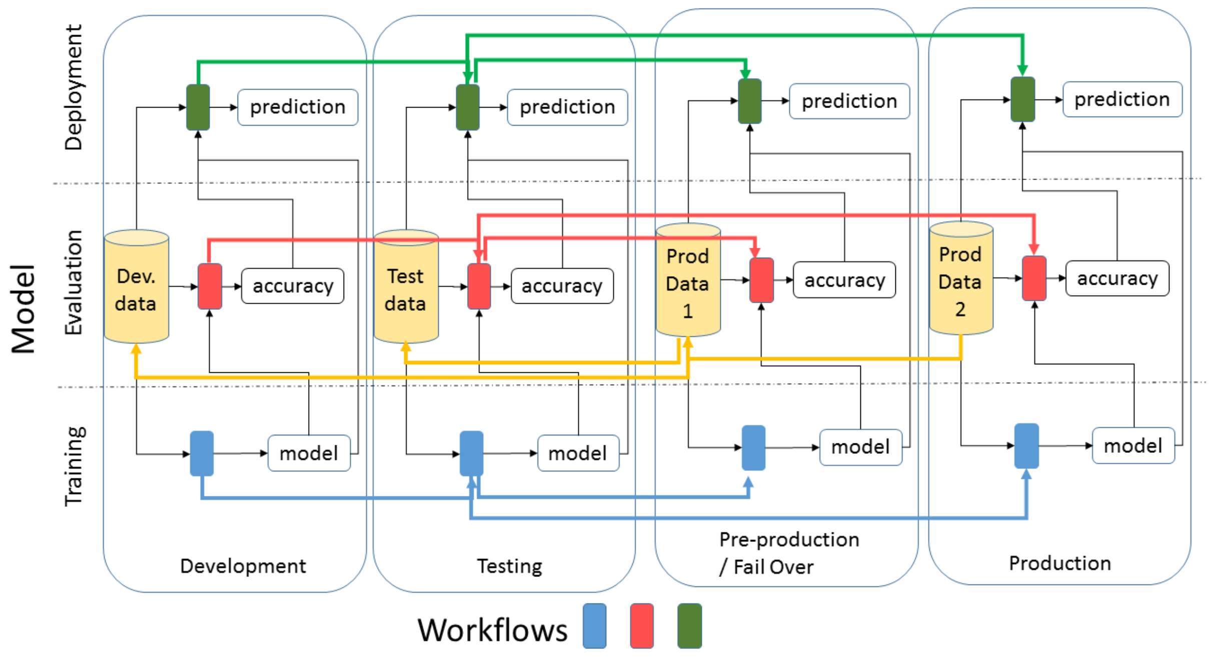
Some nodes, among many others, that might be of help when automating workflow testing could be:

- Send Email, to send an email with testing results
- Time Difference, for speed testing
- Timer Info and Global Timer Info, to collect statistics about individual and overall execution of workflows

Besides the **KNIME Testing Framework** you also need a test data set in order to create and run test cases. If the development data set is sufficiently general and updated frequently enough, this could also be used as a test data set. Otherwise a special data set has to be created for testing and maintained regularly. This data set needs to cover all possible data constellations, from the most frequent and intuitive to the rarest and weirdest ones.

Figure 3 shows the full machine and data architecture, with the testing, development, and pre-production environments as replicas on different scales of the production environment.

Figure 3. Full Machine Architecture with Development, Testing, Production, and Fall Back Environments for all workflows implementing Model Training, Model Testing, and Model Deployment. Workflows move from Development to Testing, to Fall back/Fail over, and to Production. Fall-back Data are a copy of Productive Data. Data for Development and Testing are a Subset of Fall-back (= Productive) Data.



The cycle in figure 3 starts with the development environment when a new workflow is created. For every new feature added to the workflow during development, a test case has to be designed and added to the testing environment. After development, the workflow is then moved into the testing area to be thoroughly tested. If the workflow passes all test cases, it is considered ready for release and can be moved into the production and fall-back area.

Sizing the Machines

Sizing the machines for each environment is more of an art than a science. Too much depends on external factors like data amount, response requirements, processing mode (batch vs. streaming), etc. There are no rules, for example that "you need so many GB of RAM for every TB of storage".

Typically, **I/O** is the bottleneck, so more RAM and more and faster disks are always a good idea. Only when you saturate your internal bus with data, you will need to think about faster CPUs.

In order to increase the amount of **memory** available to KNIME Analytics Platform, you can change the –Xmx parameter in the knime.ini file to 16G for example, providing you have more than 16G RAM available on your machine.

In order to increase the execution speed of a single node, you can enable the option "Keep all in memory" in the Memory Policy tab of the configuration window of each node. This option forces all input data into memory for execution. The only caveat here is to check whether your machine can handle such a memory load.

Roles in a Data Science Lab

Working in the environments described above requires several roles, which, of course, can be played by a single person. However each role requires a different mindset and access rights.

The **administrator** is responsible for setting up and configuring the machines, the environments, the repositories, as well as monitoring the data update and exchange. For each environment, repository, and machine, he/she needs to set up user permissions, backups, and the process schedule.

The **workflow developer** creates, updates, and corrects workflows and corresponding initial test data sets, committing them to the workflow and the data repository respectively. She/he is also responsible for the creation and maintenance of possible metanode templates.

The **workflow tester** verifies the correct functioning of the workflows, logic- and execution-wise. He/she is responsible to expand the original test set to cover increasing data occurrences – from standard occurrences to infrequent unintuitive occurrences and to develop the test procedures. Test cases range from unit tests and complete test sets through to stress test sets.

The **data scientist** builds, evaluates, and monitors BI reports and predictive models, using ad hoc developed workflows or regularly scheduled workflows.

The **operator** deploys the workflows and models in production, once she/he gets the "go" signal from the tester and the data scientist. She/he needs to make sure that workflows and models are run/used according to the design specs.

Conclusions

We have tried here to provide a few guidelines based on our experience about the DWH infrastructure needed for a data science lab: environment segregation, testing, data sets, role management, user permission, resource sharing, user authentication, rollover, versioning, dashboards, and more features needed in a modern data science lab.

References

1. Cross- Industry Standard Process for Data Mining - CRISP-DM Process definition on Wikipedia http://en.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining
2. KNIME Analytics Platform www.knime.org