CLOUDERA MACHINE LEARNING: CAPABILITIES AND APPROACH FOR AI, AT SCALE, IN THE ENTERPRISE

PREPARED BY
BLUE BADGE INSIGHTS, INC.

BLUE BADGE

WWW.BLUEBADGEINSIGHTS.COM
OCTOBER 2020

TABLE OF CONTENTS

INTRODUCTION

Enterprise organizations throughout the globe are pursuing their digital transformations, an imperative accelerated by the worldwide pandemic and the added emphasis of distanced, online customer interaction that comes with it. Being on the vanguard of digital transformation, AI and machine learning (ML) are at the top of their hype curve. Pre-pandemic interest in AI was high; but, given current digital business requirements, it has reached the stratosphere.

While such attention on AI is creating a lot of excitement and driving great innovation, it can also make it difficult for customers to implement the technology successfully. Cutting through marketing “noise” when a sector is hype-driven can be a formidable challenge. It can also push vendors to focus on features that demo well and are high on “cool” factor, rather than those that address true pain points and are more applicable to successful implementation.

Machine learning operations (MLOps) functionality bucks this trend of noise and superficiality. Instead, it supports customers’ ability to operationalize their machine learning efforts, helping them to implement successfully at scale, and proactively manage ongoing use of machine learning models, not just initial deployments. MLOps can manage model experimentation, deployment, monitoring and more. In various ways, it can also help customers stay on top of keeping their models accurate and, thereby, fair.

Cloudera Machine Learning (CML) has sophisticated MLOps functionality integrated into the platform, in a fashion that provides an unusual balance of the sometimes-dueling strong suits of structure and flexibility. As part of the overall Cloudera Data Platform (CDP) — Cloudera’s hybrid cloud-native enterprise data platform — CML enables end-to-end machine learning in the context of a comprehensive data management, analytics and data engineering platform, ensuring MLOps integration into the broader data practices and operations of an organization. The major aspects of CML’s value proposition and capability set are summarized in Figure 1.

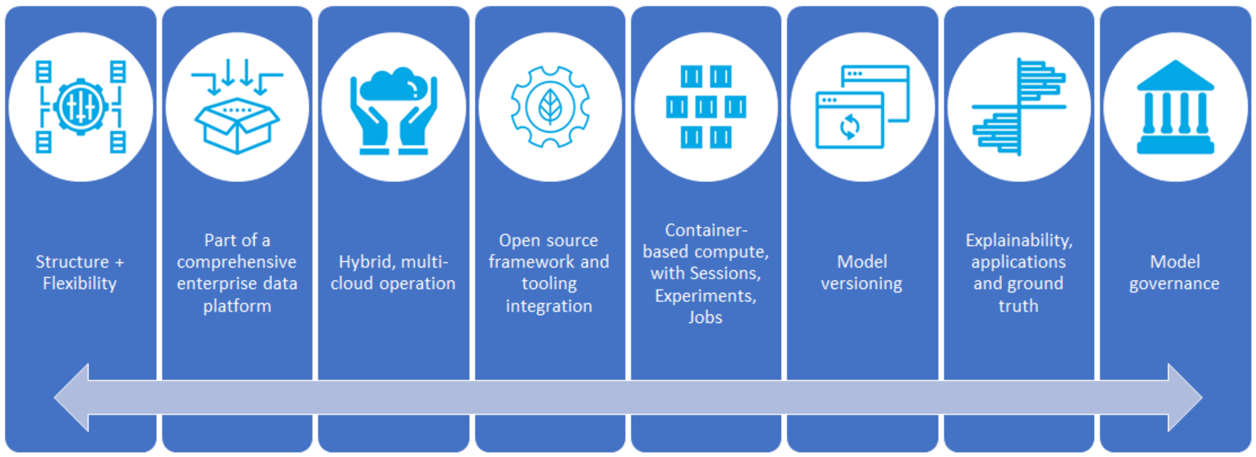

Structure + Flexibility
Part of a comprehensive enterprise data platform
Hybrid, multi-cloud operation
Open source framework and tooling integration
Container-based compute, with Sessions, Experiments, Jobs
Model versioning
Explainability, applications and ground truth
Model governance

Figure 1: The Cloudera Machine Learning Value Proposition - A full lifecycle production machine learning platform for quickly and securely going from data, to experimentation, to production of machine learning models at scale.

This report will provide an examination of CML's MLOps capabilities. But rather than function only as an explainer, the features will be discussed in the context of how they address industry and customer needs. This is important, and not just a gimmick: since CML's MLOps features were primarily driven by customer requirements and requests, such a contextual exploration will provide the full story of why the functionality exists and why it was prioritized. This is the best way to judge the efficacy and utility of these capabilities and assess their value. With that said, let's proceed.

CML OVERVIEW

Cloudera Machine Learning is impressive in its own right, but has the great advantage of running as part of the Cloudera Data Platform. CDP is a hybrid cloud-native enterprise platform for data science, analytics, streaming data, data engineering and data management. It provides the ability to spin up managed and secure environments with a data hub and data lake for managing data, access, and security; as well as the ability to deploy data engineering clusters, machine learning workspaces, and data warehouse infrastructure on a just-in-time basis. CDP's hybrid data architecture enables you to run it in your own data center, private cloud, the Microsoft Azure and Amazon Web Services public clouds, or a combination of these. Better yet, it offers its own Shared Data eXperience (SDX) platform for managing these distributed assets together, with shared data, security and centralized data governance. The CDP home screen is shown in Figure 2, with the Cloudera Machine Learning tile highlighted.

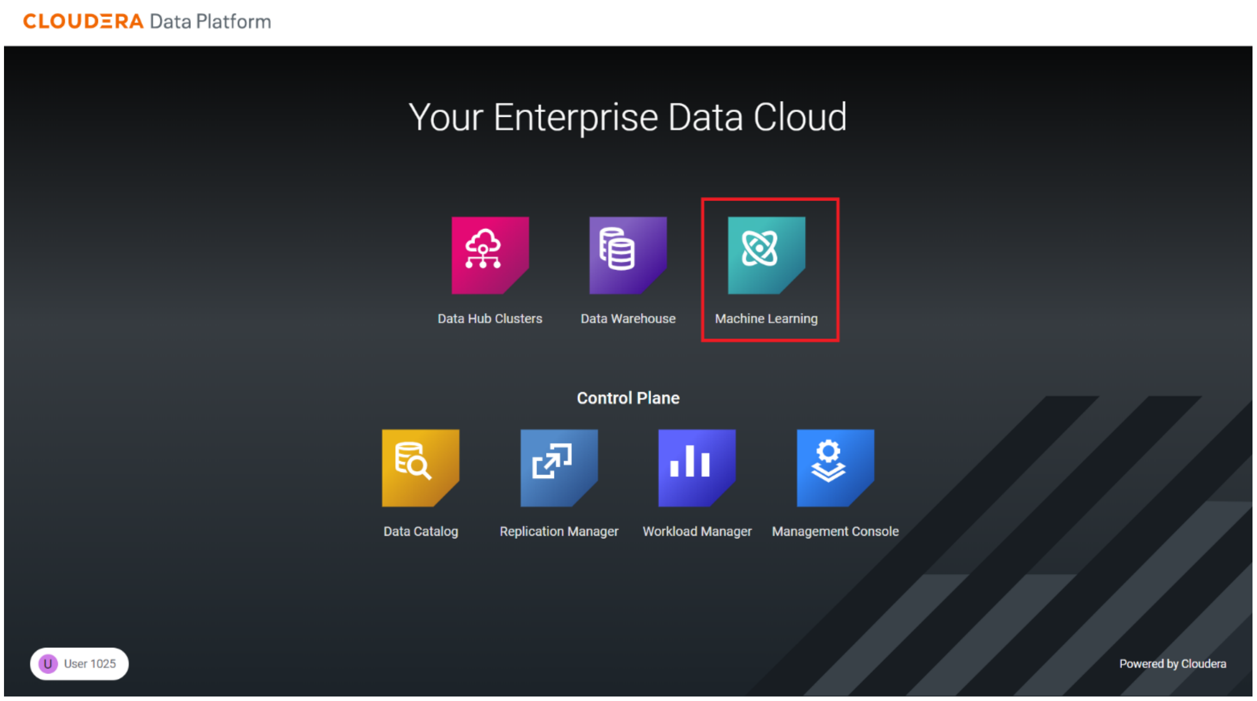


Figure 2: The CDP home screen, with CML installed

CML operates as a first-class citizen within the CDP control plane, alongside the other core components the latter manages and controls. CML also integrates with CDP's data governance framework, ensuring machine learning processes and users participate in the same data catalog and lineage infrastructure as do conventional data sources and data sets. This is, and should seem, perfectly logical and intuitive. But it's actually unusual, as many AI platforms function in a standalone fashion, creating silos of functionality and making it difficult for organizations to govern their data and machine learning models in a unified, coordinated fashion.

OPEN SOURCE

Much of CDP is based on Apache Hadoop, Apache Spark and other open source projects in the big data ecosystem. Spark's support for SQL-based querying and analysis, and its inclusion of the Spark MLlib set of machine learning algorithms, are immediately available to all CML developers, in Python, R or Scala. The very core of CML is based on well-known, open source execution platforms, programming languages and frameworks.

But developers can go well beyond the core and utilize other widely-adopted, open source data science frameworks. These would include Python-specific libraries like Numpy, Pandas, Scikit-learn, Seaborn, and Matplotlib; deep learning frameworks like PyTorch and TensorFlow; and model explanation frameworks like LIME and SHAP. None of these libraries is required, but any and all of them are fully accommodated, allowing them to be stitched into what will become a customer's or team's own core platform.

Depending on the network configuration, CML can load any open source project dependencies it needs from the Python Package Index (PyPI) by using the pip package management utility from either the public repository or a copy hosted locally on a secure network. In the next section of this report, we'll discuss how CML manages scripts in its projects, but for now, know that creating a simple bootstrap script in CML that, among other things, contains the pip3 commands necessary for the project, is a good way to automate the loading of dependent packages. The build script can also import necessary code package namespaces, set up global environment variables, provision cloud storage and upload the necessary raw data to it.

STRUCTURED FLEXIBILITY

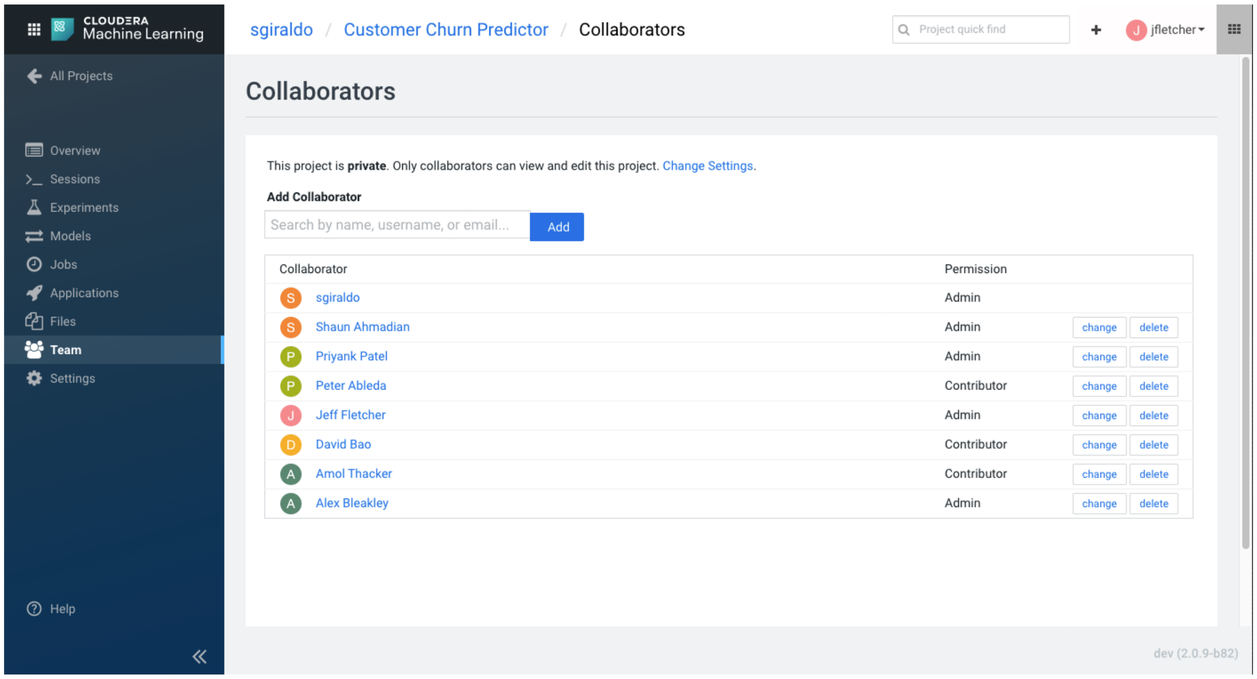
Such a use of scripts to automate certain tasks is a good example of combining code with CML's UI-driven features. As it turns out, this blended approach is a hallmark of CML, and it's a design point that is quite intentional.

We've mentioned already how many ML platforms operate in a standalone fashion, and how that creates certain challenges in making ML work as part of the mainstream technology efforts in an organization. Part of the reason ML platforms are functionally and operationally segregated is because data science teams have often been organized separately from mainstream analytics and software development teams, and have developed distinct toolchains, standards and even culture.

In order for enterprises to operationalize machine learning successfully, data science teams must work in greater collaboration and coordination with their peer technology teams. That said, this cannot be imposed by fiat. Compelling data science/ML teams to use unfamiliar tools, procedures and structures would be too abrupt and, perhaps more important, would likely backfire. Instead, a balance must be found, so that data scientists can work the way they're comfortable working and at the same time conform to broader organizational standards and requirements.

CML avoids rigidity around tooling and practices, and instead establishes a baseline environment and constructs. A set of core organizational structures and critical entities establish this baseline, yet discretionary components and technology can be layered on top of them. For data science in the enterprise, this is key. A successful platform for modern machine learning development must provide sufficient flexibility for data scientists and ML engineers to buy into it and enjoy working with it. But it must also put those professionals and their teams into compliance with corporate security and governance requirements, without causing them to lose agility.

This is more than just philosophy; it manifests in concrete structures in CML. For example, CML customers can define shared Workspaces, domiciled in specific CDP environments, where teams can collaborate. Within a workspace, one or potentially more engine profiles (hardware profiles that define CPU, GPU and RAM allocations) can be defined. Within these workspaces, one or (usually) more Projects can be created by individual Users. These projects can be private, shared with other specific users, or with formally-defined Teams that define specific users as collaborators. This is shown in Figure 3.



sgiraldo / Customer Churn Predictor / Collaborators

Collaborators

This project is **private**. Only collaborators can view and edit this project. Change Settings.

Add Collaborator

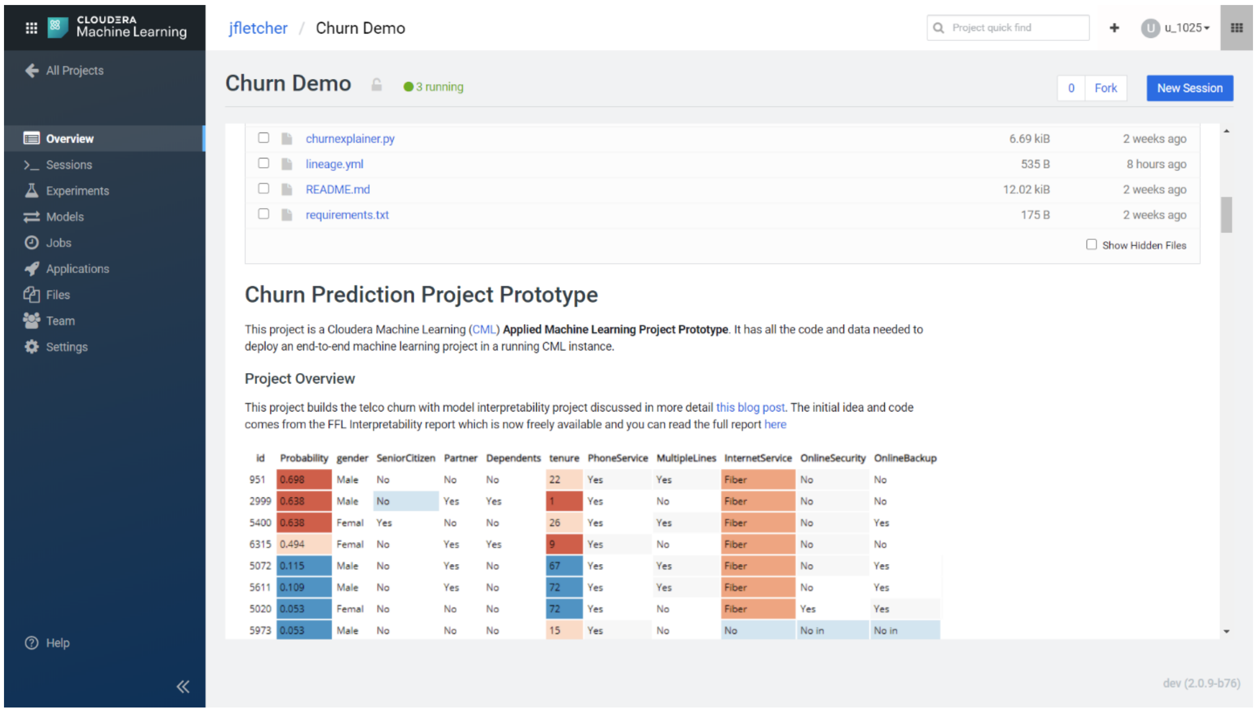
Search by name, username, or email... Add

Collaborator	Permission		
sgiraldo	Admin		
Shaun Ahmadian	Admin	change	delete
Priyank Patel	Admin	change	delete
Peter Ableda	Contributor	change	delete
Jeff Fletcher	Admin	change	delete
David Bao	Contributor	change	delete
Amol Thacker	Contributor	change	delete
Alex Bleakley	Admin	change	delete

Figure 3: CML allows collaborating Teams to be defined

Lots of what data science is about these days is the code itself. To that end, CML Projects contain Files – code scripts, first and foremost, but including any other file type as well – optionally organized within folders. CML projects also contain interactive Sessions where scripts and other interactive computing can execute interactively; Models, naturally; Experiments that can be used to train and test those models; deployments to host the models and serve as callable endpoints for scoring/inferencing against them; and Jobs, wherein specific script functions can be executed imperatively or on a scheduled basis.

A CML Project has an Overview page that includes information about the Models, Files and Jobs in the Project. It can also contain summarized documentation in markdown format if the project contains a README.md file, a feature similar to github projects. Figure 4 shows the Overview page, scrolled down to the end of the Files listing and the beginning of the documentation. Note the markdown contains a screenshot of the project's test harness application, which we will discuss in-depth, later in the report.



churnexplainer.py	6.69 kiB	2 weeks ago
lineage.yml	535 B	8 hours ago
README.md	12.02 kiB	2 weeks ago
requirements.txt	175 B	2 weeks ago

Churn Prediction Project Prototype

This project is a Cloudera Machine Learning (CML) **Applied Machine Learning Project Prototype**. It has all the code and data needed to deploy an end-to-end machine learning project in a running CML instance.

Project Overview

This project builds the telco churn with model interpretability project discussed in more detail this blog post. The initial idea and code comes from the FFL Interpretability report which is now freely available and you can read the full report here

id	Probability	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup
951	0.698	Male	No	No	No	22	Yes	Yes	Fiber	No	No
2999	0.638	Male	No	Yes	Yes	1	Yes	No	Fiber	No	No
5400	0.638	Femal	Yes	No	No	26	Yes	Yes	Fiber	No	Yes
6315	0.494	Femal	No	Yes	Yes	9	Yes	No	Fiber	No	No
5072	0.115	Male	No	Yes	No	67	Yes	Yes	Fiber	No	Yes
5611	0.109	Male	No	Yes	No	72	Yes	Yes	Fiber	No	Yes
5020	0.053	Femal	No	No	No	72	Yes	No	Fiber	Yes	Yes
5973	0.053	Male	No	No	No	15	Yes	No	No	No in	No in

Figure 4: CML Project Overview page

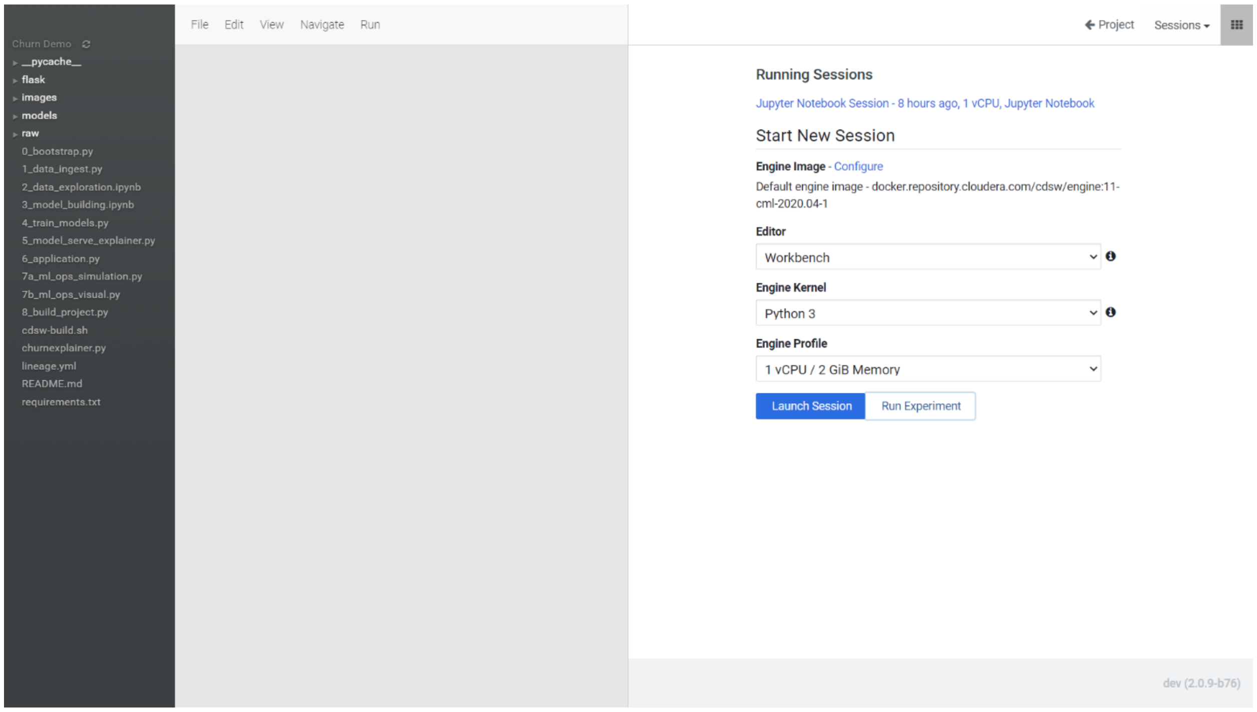
Formal documentation of this sort is important. While source code control systems have long supported this type of documentation through the addition of a markdown file in the code repository, CML's support of it through an explicit platform feature is an important facet in the effort to add rigor and process to ML development.

Speaking of source code control, CML projects can be integrated with any Git-compliant repository, perhaps the most popular source code control standard, to provide version control at the source code/script level, further illustrating the congruity between ML development and mainstream software development. When you create a CML Project, you can optionally supply a Git URL that points to any accessible repository. While CML does not provide direct UI support for Git, its commands can be run from within the Workbench terminal (described in the next section).

CML also accommodates versioning for models. This support, which is described later in this report, is provided directly by CML and not dependent on Git.

SESSION-BASED COMPUTE

We'll discuss later in this report how Experiment- and Job-based compute can be provisioned to run specific, discrete, short-lived tasks. But for the significant amount of interactive work the ML development process involves, CML supports the concept of compute Sessions. Sessions involve on-demand spinning up of compute resources, based on specific language kernels and engine profiles, with their own processing and memory parameters. Figure 5 shows the CML experience for starting a session.



Running Sessions

Jupyter Notebook Session - 8 hours ago, 1 vCPU, Jupyter Notebook

Start New Session

Engine Image - Configure

Default engine image - docker.repository.cloudera.com/cdsw/engine:11-cml-2020.04-1

Editor: Workbench

Engine Kernel: Python 3

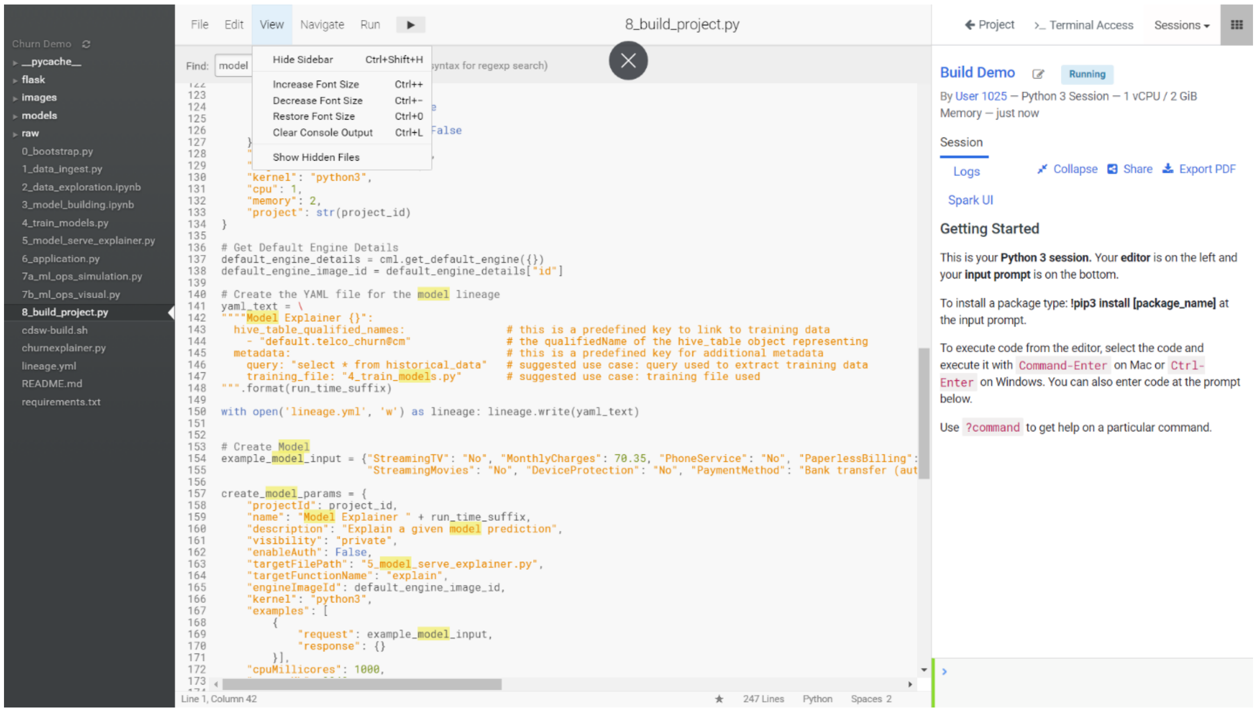
Engine Profile: 1 vCPU / 2 GiB Memory

Figure 5: Launching a new CML Session, with specification of editor environment, kernel and engine profile

CML WORKBENCH

As the preceding figure shows, provisioned Sessions also specify an associated editor environment. In CML, data scientists can create and run interactive code within Jupyter notebooks or from their own preferred integrated development environments (IDEs). But CML also boasts its own Workbench, which includes a powerful native editor with syntax highlighting, and an output console, where resulting messages, text, markdown and visualizations produced by the code can be rendered.

The Workbench thus provides most of the same interactive capabilities of a notebook while fostering the authoring of full-fledged standalone scripts that can be run in any language-compatible environment, including the command line. The Workbench editor is shown in Figure 6.



```
    "kernel": "python3",
    "cpu": 1,
    "memory": 2,
    "project": str(project_id)
}

# Get Default Engine Details
default_engine_details = cml.get_default_engine({})
default_engine_image_id = default_engine_details["id"]

# Create the YAML file for the model lineage
yaml_text = \
"""Model Explainer {}":
  hive_table_qualified_names:                 # this is a predefined key to link to training data
    - "default.telco_churn@cm"                # the qualifiedName of the hive_table object representing
  metadata:                                   # this is a predefined key for additional metadata
    query: "select * from historical_data"    # suggested use case: query used to extract training data
    training_file: "4_train_models.py"        # suggested use case: training file used
""".format(run_time_suffix)

with open('lineage.yml', 'w') as lineage: lineage.write(yaml_text)


# Create Model
example_model_input = {"StreamingTV": "No", "MonthlyCharges": 70.35, "PhoneService": "No", "PaperlessBilling":
                       "StreamingMovies": "No", "DeviceProtection": "No", "PaymentMethod": "Bank transfer (aut

create_model_params = {
    "projectId": project_id,
    "name": "Model Explainer " + run_time_suffix,
    "description": "Explain a given model prediction",
    "visibility": "private",
    "enableAuth": False,
    "targetFilePath": "5_model_serve_explainer.py",
    "targetFunctionName": "explain",
    "engineImageId": default_engine_image_id,
    "kernel": "python3",
    "examples": [
        {
            "request": example_model_input,
            "response": {}
        }],
    "cpuMillicores": 1000,
```

By User 1025 — Python 3 Session — 1 vCPU / 2 GiB Memory — just now

Getting Started

This is your **Python 3 session.** Your **editor** is on the left and your **input prompt** is on the bottom.

To install a package type: **!pip3 install [package_name]** at the input prompt.

To execute code from the editor, select the code and execute it with `Command-Enter` on Mac or `Ctrl-Enter` on Windows. You can also enter code at the prompt below.

Use `?command` to get help on a particular command.

Figure 6: The CML Workbench and its code editor

The Workbench also provides its own interactive command prompt (shown at bottom-right in the preceding figure) and web-based terminal access (not shown), both of which execute in the context of the underlying Session. Scripts that are authored and executed interactively in the Workbench can further serve as the basis for the Experiments, Models and Jobs mentioned earlier, and described in the next section of this report.

Again, a balance is struck: customers can use the full Workbench capabilities with the rest of CML and thus avoid having to stitch together additional dev tools in a bespoke fashion. At the same time, if the data science and/or ML engineering teams have already adopted certain dev tools, those can be accommodated and integrated. This creates a big tent, allowing CML to be the nucleus for ML work – at the very least – and, for those customers who want more – to serve as the sanctioned, turn-key ML environment.

CONTAINERS, KUBERNETES AND PROVISIONING OF COMPUTE

Everything in terms of compute in CML is based on Kubernetes (K8s). The workspace itself defines a K8s auto-scaling cluster, while deployments are container images that can be deployed within it. Sessions are pods within the workspace cluster, based on the engine definitions already discussed. Cloudera provides the base docker images used for these Sessions, which can be extended to add libraries and code specific to your team's requirements. Experiments are also pods based on engine definitions, but whereas Sessions are for interactive compute and run until explicitly shut down, Experiments' lifespans are task-driven, being spun-up and torn-down to execute a specific script that trains and tests a model, then optionally records specific metrics based on both.

Such a container-based architecture is state of the art. It's also just plain smart. On the other hand, the rigors of working with containers and orchestrating them with K8s are beyond scope for data science and ML engineering work. Forcing data scientists and ML engineers to become Kubernetes experts creates an unreasonable burden on them and deep inefficiency for the organization. The beauty of CML is that it performs all the K8s work behind the scenes, ensuring users are utilizing a proper, modern architecture and yet relieving them of the significant learning curve and management overhead of having to deal with K8s themselves.

PROJECT TEMPLATES AND WORKFLOWS

While some structures are fixed and concrete, others are optional and based on convention. Specifically, Projects can be based on Cloudera's Applied Machine Learning Prototype templates which can define or imply a specific workflow. CML's default Churn Predictor template, for example, contains script and notebook stubs to accommodate each of several discrete workflow steps. CML doesn't mandate those steps, nor a particular sequence of executing them. But it does make doing so easy, and it helps enforce adherence to that (or another) methodology, should an organization wish to standardize on it.

THE CANONICAL WORKFLOW

The "canonical" workflow implied by CML's built-in Churn Predictor template involves steps for ingest of data; data exploration and transformation; training, testing and deployment of models; running predictions against the models in production, and explanations of those predictions; and management tasks including monitoring models for accuracy and performance, integration of ground truth and model governance.

Again, this workflow isn't mandatory. But it does represent a set of best practices, and discussing each step within it is a great way to get a full sense of how CML works. For the remainder of this report, we'll go through the workflow, and glean from it the unique capabilities of CML, as well as issues important to ML development work overall.

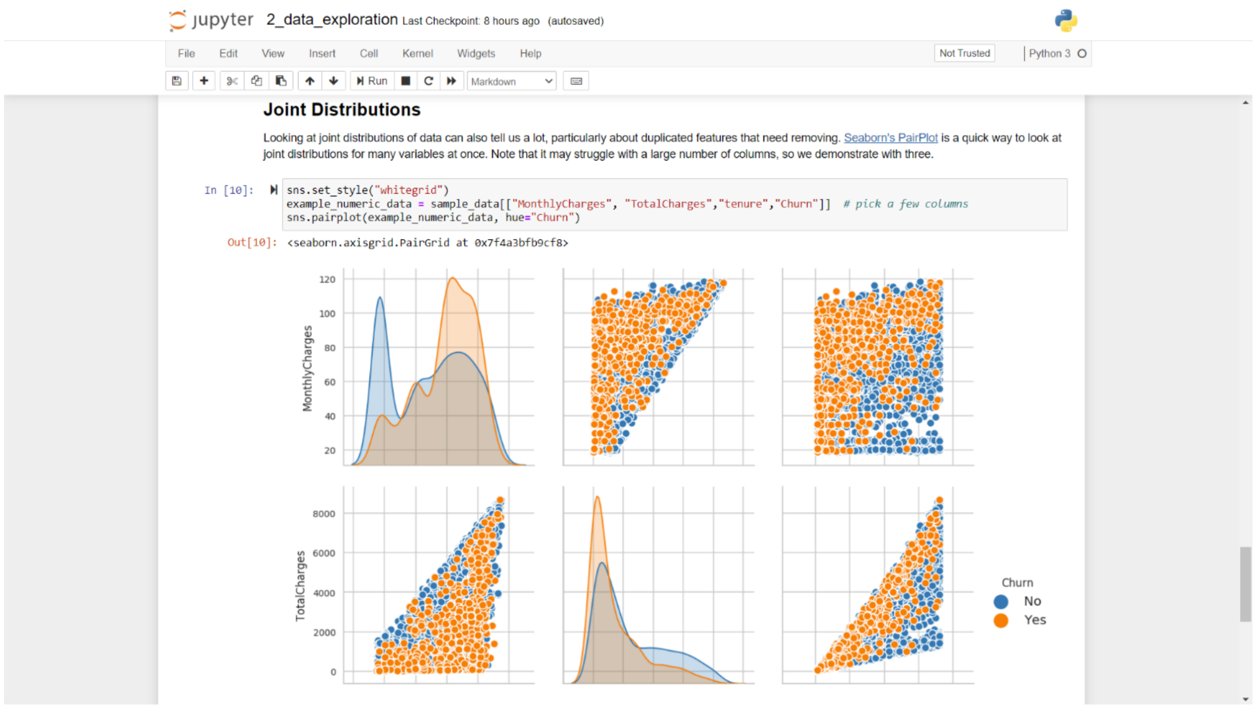
SETUP AND INGEST

Data ingest is likely to be the most straightforward step in the ML development workflow, as it's more about straight data access than it is machine learning, per se. Typically, the code in this step will pull data back from the data lake, a data warehouse, or some public/open data set endpoint. The data will likely be loaded into a dataframe, then perhaps saved in the local CML environment in an open format, such as CSV.

DATA EXPLORATION AND TRANSFORMATION

Next, the data scientist or ML engineer may aggregate, profile, and visualize the data, with an eye toward understanding correlation between columns and beginning to determine target and feature columns. Additional code may determine data quality, then cleanse the data set to remediate any quality issues identified. Finally, feature engineering, algorithm selection and hyperparameter optimization can be performed, in order to ready things for training.

Data exploration and transformation work is the bread and butter of the pre-training data science workflow. While some of this might be performed in a script, CML also facilitates performing this work in Jupyter notebooks. The latter is shown in Figure 7.



Joint Distributions

Looking at joint distributions of data can also tell us a lot, particularly about duplicated features that need removing. Seaborn's PairPlot is a quick way to look at joint distributions for many variables at once. Note that it may struggle with a large number of columns, so we demonstrate with three.

```
In [10]:  sns.set_style("whitegrid")
          example_numeric_data = sample_data[["MonthlyCharges", "TotalCharges","tenure","Churn"]]  # pick a few columns
          sns.pairplot(example_numeric_data, hue="Churn")

Out[10]: <seaborn.axisgrid.PairGrid at 0x7f4a3bfb9cf8>
```

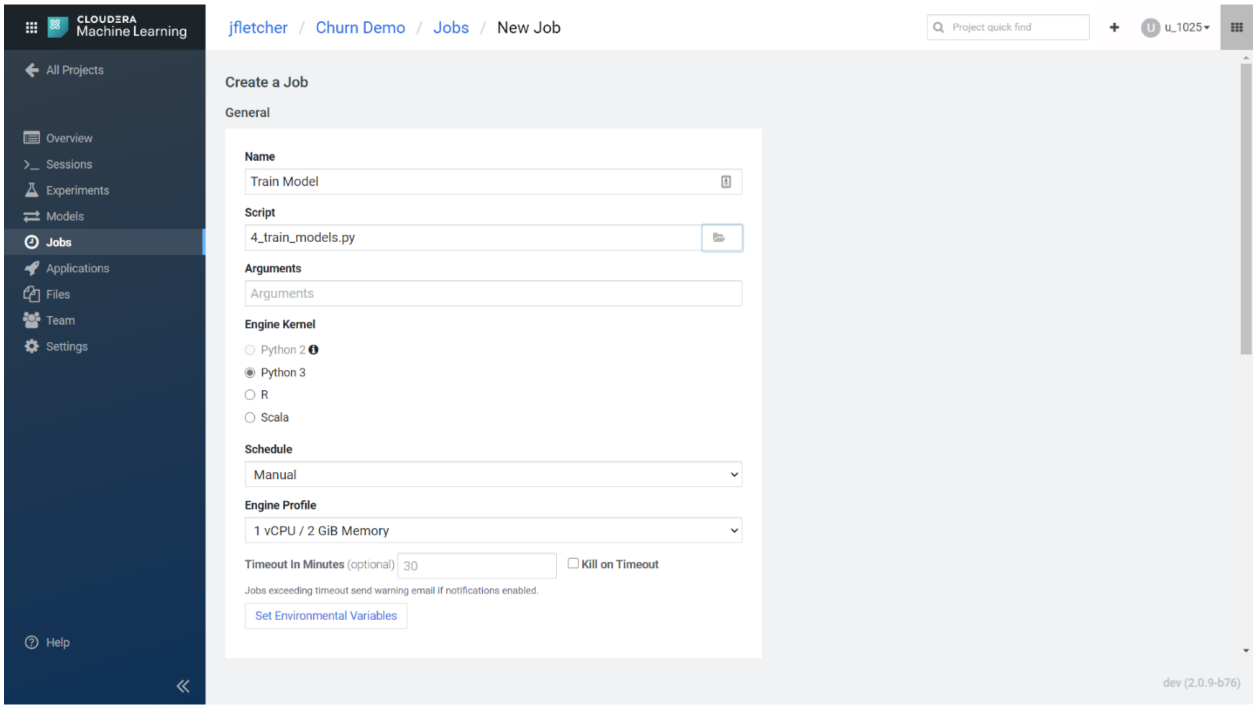
Figure 7: Data exploration in a Jupyter notebook, hosted within CML

MODEL TRAINING

Training is where the "action" takes place in ML development, as it produces the model itself. In CML, a script for this step may actually have two "identities." Like other scripts in the project, it can be run interactively in a Session to perform certain one-time tasks. On the other hand, the actual training code can be placed in a specific function within the script, which can then be called directly by a CML Job, or in a CML Experiment (discussed in the "Test the Model" section, below), to train and test the model. As such, it serves in both interactive/development and scheduled/production capacities.

The fact that CML has a built in Jobs engine is critical for enterprises who have adopted machine learning and are putting models in production at scale. In the proof-of-concept stage, it may be sufficient for models to be trained by running the script interactively and eventually deploying the model on a similar interactive basis, and many data scientists may be accustomed to such an ad hoc workflow. But such a "one-and-done" approach is not sufficient for enterprise-grade production ML work. For the latter, the model must be routinely re-trained on new data to keep it up-to-date and accurate. And, again, that accuracy must also be monitored, in a separate step, which we will discuss below.

In addition to the scheduling information, and as with CML Sessions, Jobs are configured with an engine kernel and profile. These can be identical to, or different from, the engine profile used for the interactive Session used in development; all of this is shown in Figure 8.



Create a Job

General

Name: Train Model

Script: 4_train_models.py

Arguments: Arguments

Engine Kernel: Python 2 [✓ Python 3] R Scala

Schedule: Manual

Engine Profile: 1 vCPU / 2 GiB Memory

Timeout In Minutes (optional): 30 ☐ Kill on Timeout

Jobs exceeding timeout send warning email if notifications enabled.

Set Environmental Variables

dev (2.0.9-b76)

Figure 8: CML Job creation

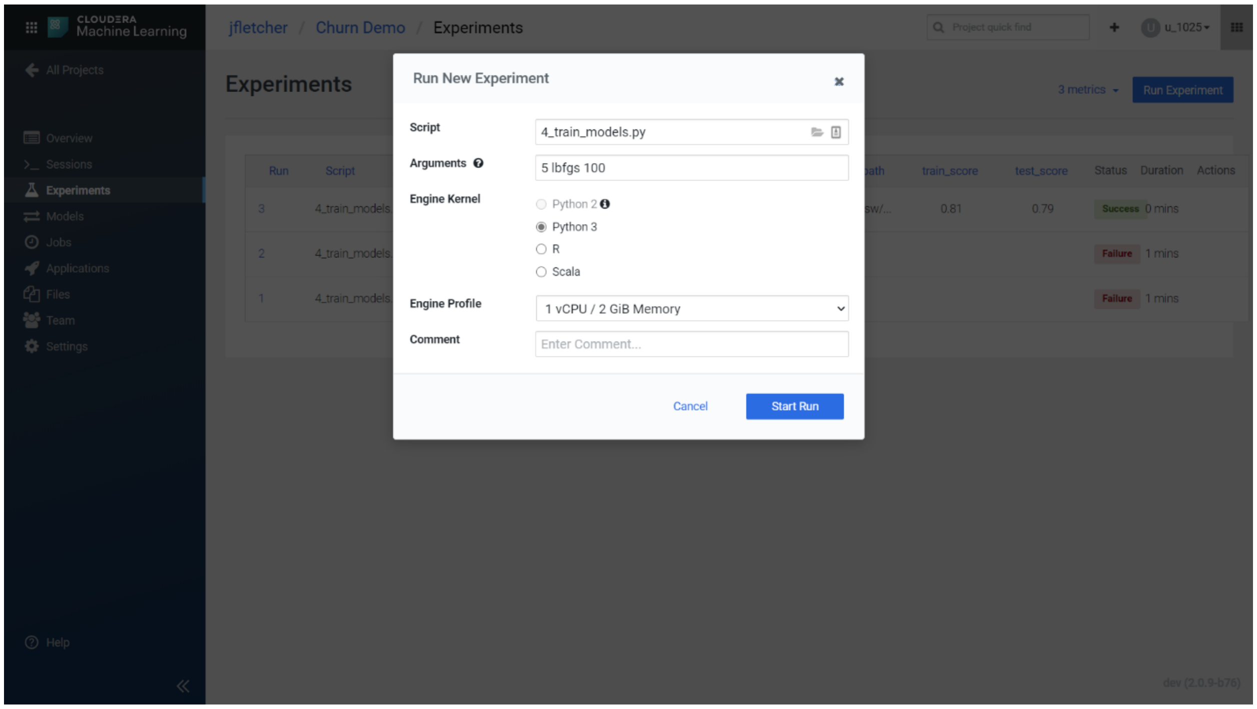
TEST THE MODEL

The dataframe used to train a model is typically a subset of the overall data set initially ingested. The remainder of the data should be used to test the model, so the model's predictions can be compared to the actual target variable in order to determine an initial accuracy level of the model. If that accuracy level is low, feature engineering and hyperparameter optimization code can be further iterated upon. The test script can then be run again to determine if the model is suitable for deployment and operationalization.

Along with such code, data scientists can use CML's special track_metric function (part of a built in python library called cdsw) to log the parameter values and the accuracy of the resulting model and the track_file function can be used to save a persisted version of the trained model.

Since optimizing the model is an iterative process, CML allows the training and testing code to be run repeatedly within a set of and parameter value-specific Experiments. Unlike Jobs, Experiments execute only on-demand; but like Jobs, they do so outside the context of a Session. As such, an engine kernel and compute/memory profile are once again supplied specifically for the Experiment.

Along with the engine profile, a script and arguments (feature values) are specified for Experiments as well, as shown in Figure 9. Combined with the aforementioned track_metric and track_file functions, this gives CML everything it needs to execute a training run, test the resulting model, record its hyperparameter values, accuracy and other metrics, and save the model file itself.



Run	Script	...path	train_score	test_score	Status	Duration	Actions
3	4_train_models...	sw/...	0.81	0.79	Success	0 mins	
2	4_train_models...				Failure	1 mins	
1	4_train_models...				Failure	1 mins	

Script: 4_train_models.py
Arguments: 5 lbfgs 100
Engine Kernel: Python 2 / [✓ Python 3] / R / Scala
Engine Profile: 1 vCPU / 2 GiB Memory
Comment: Enter Comment...

Figure 9: Running an Experiment in CML

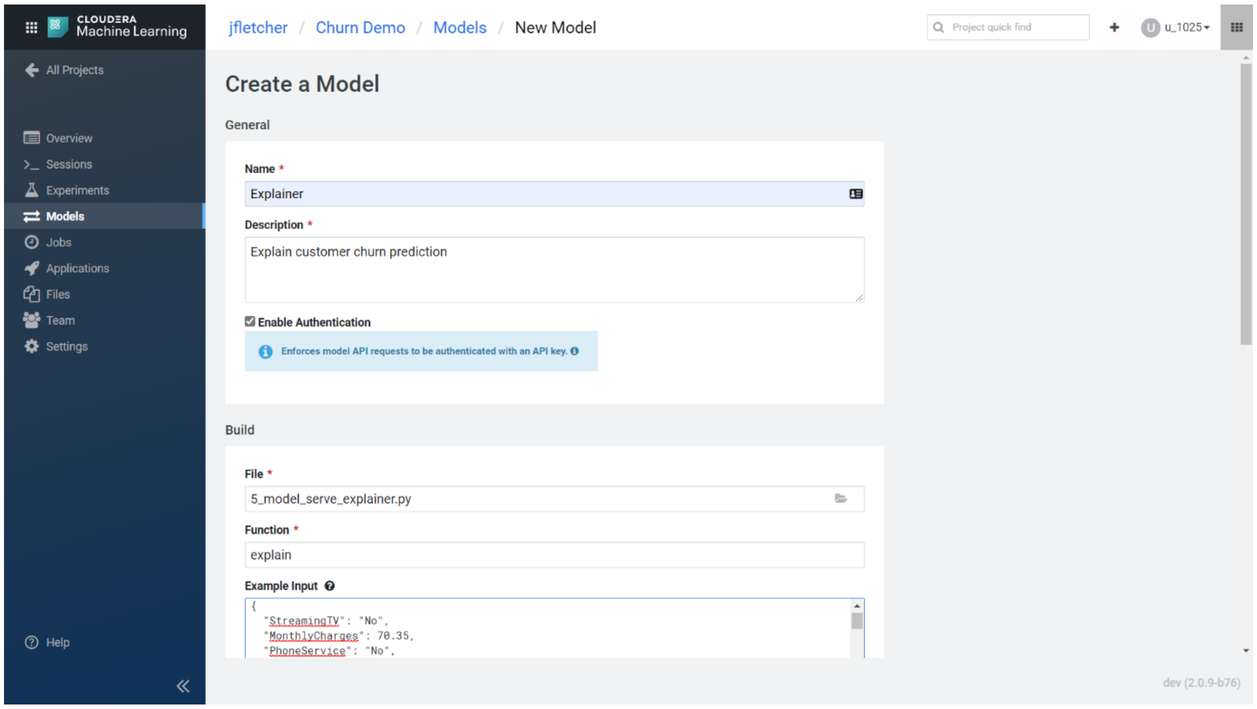
After several Experiments are run and the best one is determined, an ML engineer can go to the Overview page for that experiment, select its persisted model file right from the UI, then click Add to Project. This will make the model file available from the project's Files page for easy subsequent Model entity creation and deployment, each of which we discuss next.

MODEL CREATION

Creating a Model entity in CML allows initial and successive builds and deployments of the model to take place. Model creation is straightforward, requiring input of a name for the model, and identification of a script, and a function within the script, that will run predictions and, optionally, explanations of the predictions (discussed more later). The script, by the way, can open the persisted model file from the optimal Experiment, as discussed in the previous section.

Optionally, sample input (i.e. feature names and values) can be supplied to facilitate easy execution of a sample prediction against the model, once the model entity has been saved.

The CML Model creation screen is shown in Figure 10.



Create a Model

General

Name *
Explainer

Description *
Explain customer churn prediction

☑ Enable Authentication
Enforces model API requests to be authenticated with an API key.

Build

File *
5_model_serve_explainer.py

Function *
explain

Example Input

```
{
  "StreamingTV": "No",
  "MonthlyCharges": 70.35,
  "PhoneService": "No",
```

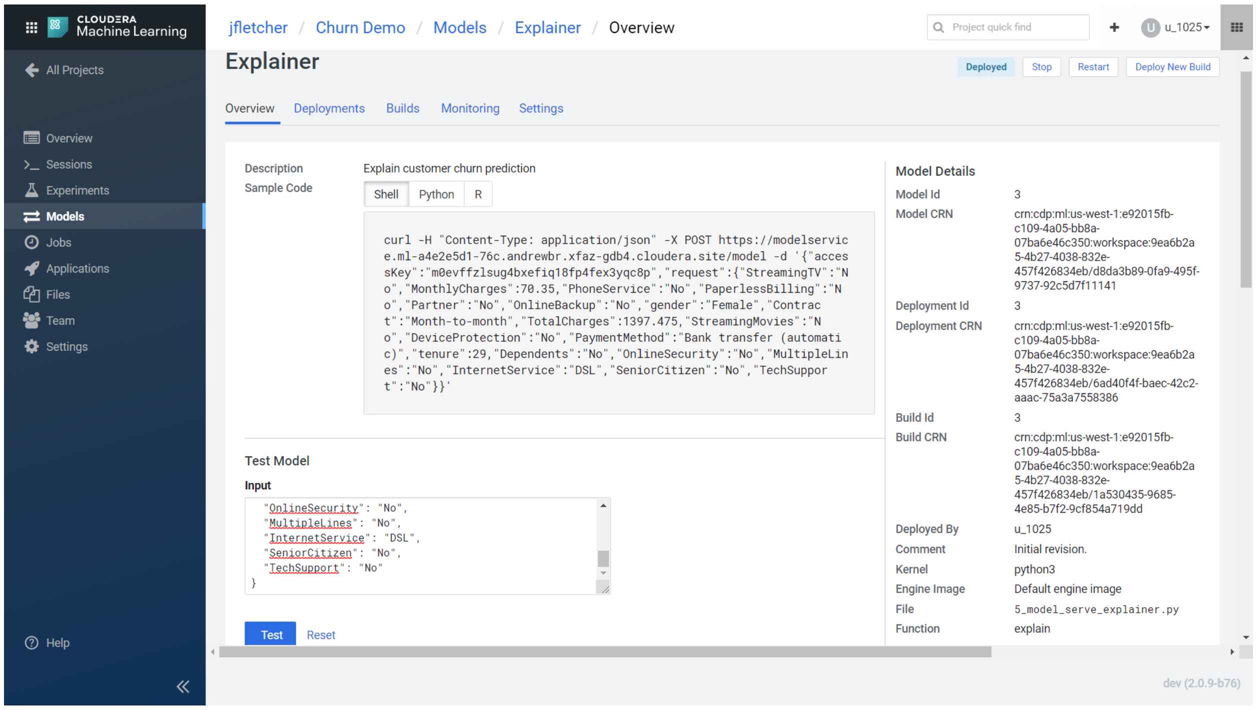
Figure 10: CML model deployment

DEPLOY

Once the CML Model is created, builds and deployments can be created against it. A deployment is essentially a Docker container that serves prediction requests on the model. Dependencies are included; CPU/GPU/RAM and number of replicas for load balancing can be specified. When the build is executed, a deployment of the model is created, for the purpose of allowing subsequent new builds to rollback to previous deployments..

This brings up an important point: version control is prerequisite in any software discipline, and that includes machine learning. The ad hoc data science approach of reverting to an old version of a notebook, then manually re-training and re-deploying a model, isn't good enough. AI platforms need to support a single-click mechanism for model version rollback, in a way that mitigates human error, especially under high-pressure circumstances. CML provides this.

Within the Model screen, separate tabs will appear that provide a Model overview, show deployments, show builds, provide operational monitoring information, and display certain settings for the Model entity. The Overview tab is shown in Figure 11.



Explainer

Description	Explain customer churn prediction
Sample Code	

```
curl -H "Content-Type: application/json" -X POST https://modelservic
e.ml-a4e2e5d1-76c.andrewbr.xfaz-gdb4.cloudera.site/model -d '{"acces
sKey":"m0evffzlsug4bxefiq18fp4fex3yqc8p","request":{"StreamingTV":"N
o","MonthlyCharges":70.35,"PhoneService":"No","PaperlessBilling":"N
o","Partner":"No","OnlineBackup":"No","gender":"Female","Contrac
t":"Month-to-month","TotalCharges":1397.475,"StreamingMovies":"N
o","DeviceProtection":"No","PaymentMethod":"Bank transfer (automati
c)","tenure":29,"Dependents":"No","OnlineSecurity":"No","MultipleLin
es":"No","InternetService":"DSL","SeniorCitizen":"No","TechSuppor
t":"No"}}'
```

Test Model

Input

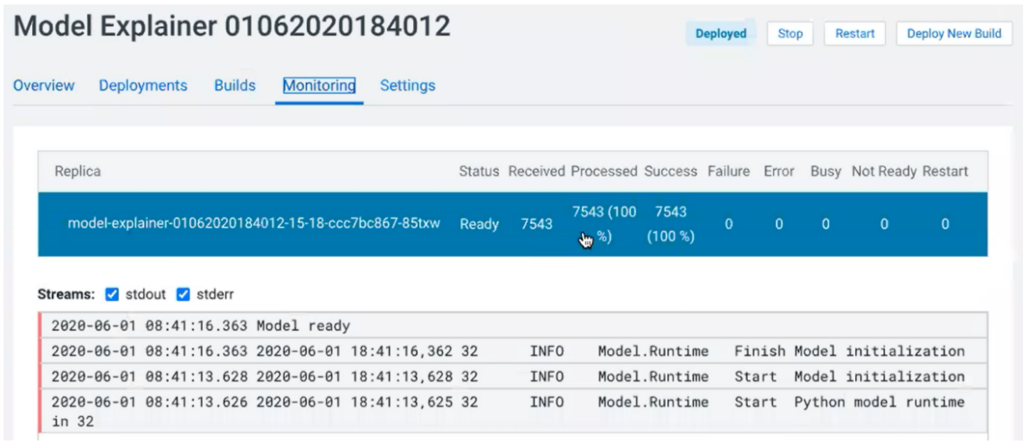
```
  "OnlineSecurity": "No",
  "MultipleLines": "No",
  "InternetService": "DSL",
  "SeniorCitizen": "No",
  "TechSupport": "No"
}
```

Model Details

Model Id	3
Model CRN	crn:cdp:ml:us-west-1:e92015fb-c109-4a05-bb8a-07ba6e46c350:workspace:9ea6b2a5-4b27-4038-832e-457f426834eb/d8da3b89-0fa9-495f-9737-92c5d7f11141
Deployment Id	3
Deployment CRN	crn:cdp:ml:us-west-1:e92015fb-c109-4a05-bb8a-07ba6e46c350:workspace:9ea6b2a5-4b27-4038-832e-457f426834eb/6ad40f4f-baec-42c2-aaac-75a3a7558386
Build Id	3
Build CRN	crn:cdp:ml:us-west-1:e92015fb-c109-4a05-bb8a-07ba6e46c350:workspace:9ea6b2a5-4b27-4038-832e-457f426834eb/1a530435-9685-4e85-b7f2-9cf854a719dd
Deployed By	u_1025
Comment	Initial revision.
Kernel	python3
Engine Image	Default engine image
File	5_model_serve_explainer.py
Function	explain

Figure 11: Overview tab of model screen

The monitoring tab of the Model screen lets users observe operational details around each build/deployment replica, as shown in Figure 12.



Model Explainer 01062020184012

Replica	Status	Received	Processed	Success	Failure	Error	Busy	Not Ready	Restart
model-explainer-01062020184012-15-18-ccc7bc867-85txw	Ready	7543	7543 (100 %)	7543 (100 %)	0	0	0	0	0

Streams: ☑ stdout ☑ stderr

```
2020-06-01 08:41:16.363 Model ready
2020-06-01 08:41:16.363 2020-06-01 18:41:16,362 32      INFO    Model.Runtime   Finish Model initialization
2020-06-01 08:41:13.628 2020-06-01 18:41:13,628 32      INFO    Model.Runtime   Start  Model initialization
2020-06-01 08:41:13.626 2020-06-01 18:41:13,625 32      INFO    Model.Runtime   Start  Python model runtime
in 32
```

Figure 12: Operational monitoring for ML models

PREDICT

If you look at the Model Details section of the Overview tab, shown on the right-hand side of Figure 11, you'll notice an Id and CRN are assigned to each model. These are critical to the model governance capabilities we will discuss towards the end of this report. You'll also notice that a Test button (shown at the lower left-hand corner of Figure 11) is provided for easy execution of a sample prediction right from the CML UI. Alternatively, a shell command as well as Python and R code, are provided to execute the sample prediction from the command shell or code. Such API calls can be locked down via authentication (via the "Enable Authentication" checkbox shown in Figure 10).

Just as track_metric can be used to log hyperparameter values and accuracy, it can also be used to log feature values as a dictionary and the output value from a prediction. A timestamp, duration and prediction UUID for the prediction will be automatically tracked by CML as well.

Builds and Deployments each get their own Ids and CRNs, and each of these is logged at prediction time so that the prediction can be associated with a particular model/build/deployment combination. This comes in handy for comparison of "ground truth" data, well after prediction – something we'll cover in the Monitoring discussion shortly.

EXPLAIN

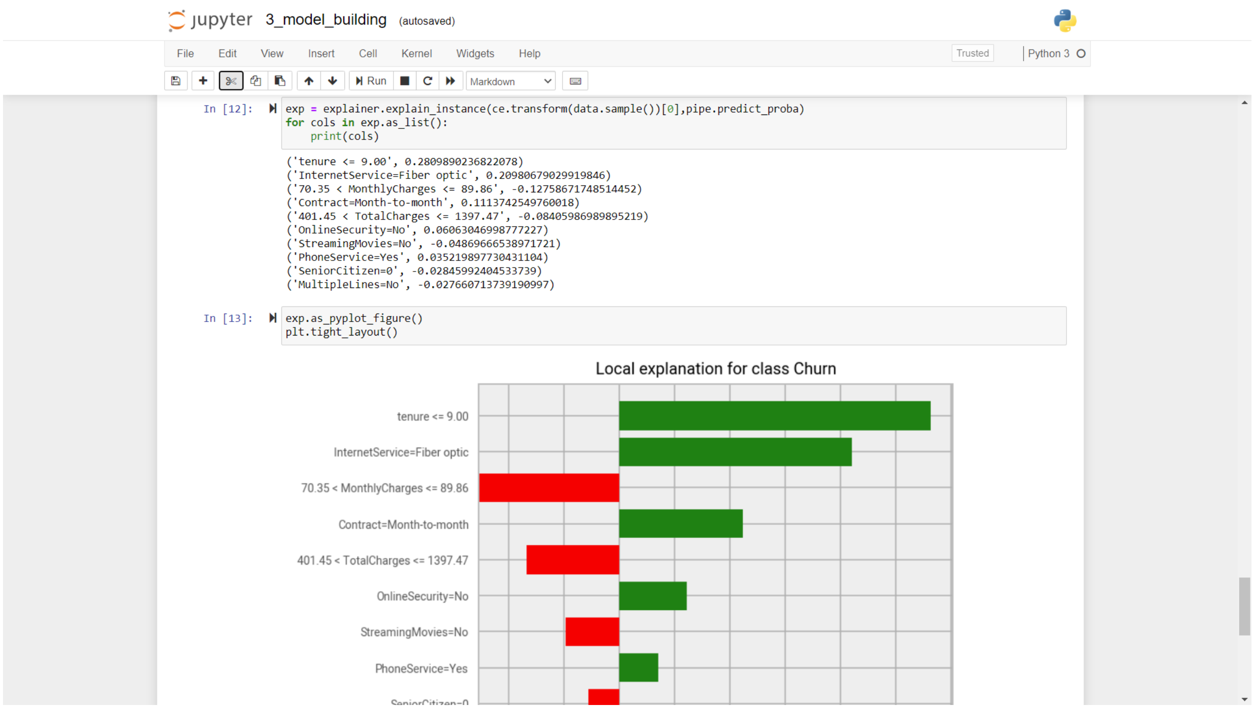
Training, testing, deploying and running predictions against a model together comprise the full arc of model development and production usage…or do they? While submitting data to a model and getting predictions back from it is absolutely where the value of ML lies, acting immediately on a model's prediction requires having a lot of faith in the model, and a comfort level with not being able to verify it. Most organizations are understandably uncomfortable putting blind faith into any technology, but with machine learning, this concern may be even more pronounced.

A prediction unto itself is valuable, but having context and an understanding for why a prediction was made will increase confidence in the model and that added confidence will likely encourage greater adoption of it. Perhaps more important, ethical considerations and a dedication to equitable machine learning requires the ability to verify the model's accuracy and understand its logic, at least empirically.

The concept of ML model explainability is still relatively new, but important open source frameworks, including LIME and SHAP, have emerged recently to help data scientists and ML engineers obtain explanations of individual predictions. While these frameworks require technical expertise, they do, within that domain, provide explanations with relative ease, and even provide ways to visualize the results.

The frameworks are still evolving, as is the explainability space in general, making it difficult to standardize on any one of them. But because CML provides that balance between built-in constructs and extensibility through integration of open source libraries, and because it also supports formalization of conventions and practices through project templates, building prediction explanations into the CML workflow is easily done. The Churn Predictor template incorporates techniques and processes discussed in the freely available Cloudera Fast Forward Interpretability report.

In fact, the default Python project template integrates LIME, and the sample model building Jupyter notebook includes the code necessary to produce and visualize an explanation, as shown in Figure 13.


jupyter 3_model_building (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
Local explanation for class Churn
tenure <= 9.00
InternetService=Fiber optic
70.35 < MonthlyCharges <= 89.86
Contract=Month-to-month
401.45 < TotalCharges <= 1397.47
OnlineSecurity=No
StreamingMovies=No
PhoneService=Yes
SeniorCitizen=0

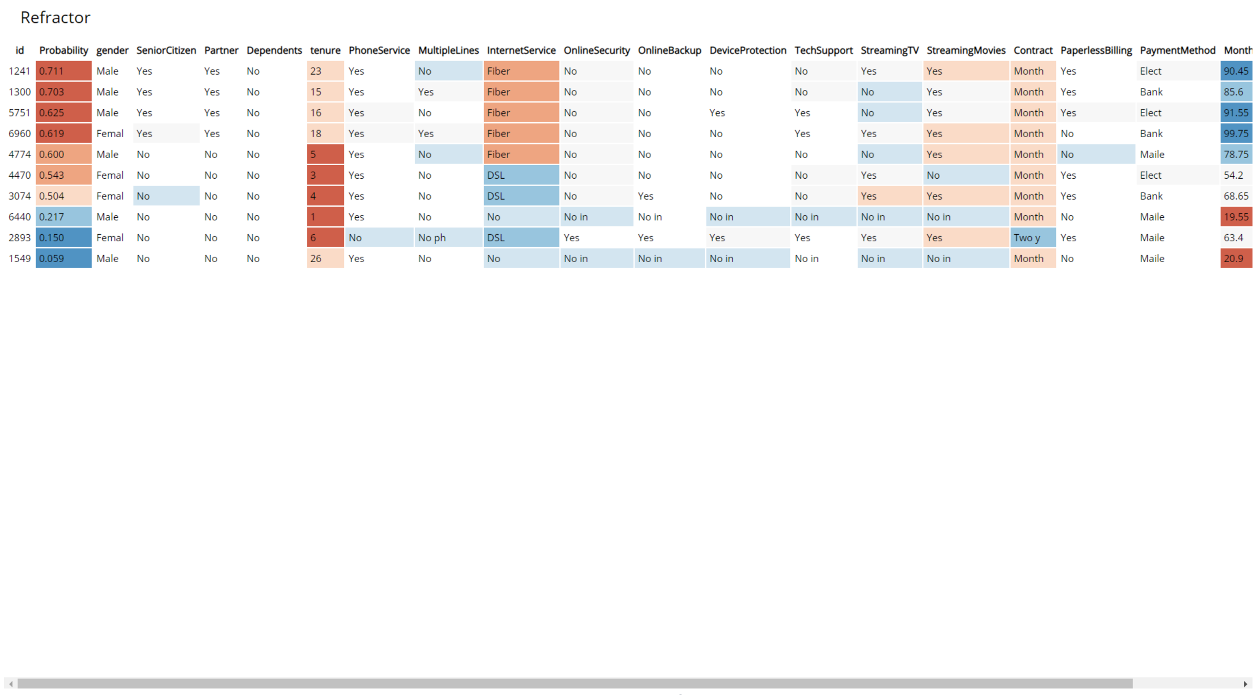
```
In [12]: exp = explainer.explain_instance(ce.transform(data.sample())[0],pipe.predict_proba)
         for cols in exp.as_list():
             print(cols)

('tenure <= 9.00', 0.2809890236822078)
('InternetService=Fiber optic', 0.20980679029919846)
('70.35 < MonthlyCharges <= 89.86', -0.12758671748514452)
('Contract=Month-to-month', 0.1113742549760018)
('401.45 < TotalCharges <= 1397.47', -0.08405986989895219)
('OnlineSecurity=No', 0.06063046998777227)
('StreamingMovies=No', -0.04869666538971721)
('PhoneService=Yes', 0.035219897730431104)
('SeniorCitizen=0', -0.02845992404533739)
('MultipleLines=No', -0.027660713739190997)

In [13]: exp.as_pyplot_figure()
         plt.tight_layout()
```

Figure 13: Model building notebook with explainer code and visualization

Note that feature values contributing to a higher predicted value for the model's target (customer churn probability in this case) are visualized as green bars extending to the right, while those contributing to lower probability are visualized as red bars extending to the left. That's one way to visualize explainability data. We discuss another approach next.

APPLICATIONS

Another way to conduct predictions is through an application, and CML Projects can be set up to generate a simple browser-based app for prediction diagnostics, automatically. The main screen of one such application is shown in Figure 14. This CML Application is a simplified version of the Refractor prototype from the Cloudera Fast Forward Interpretability report.



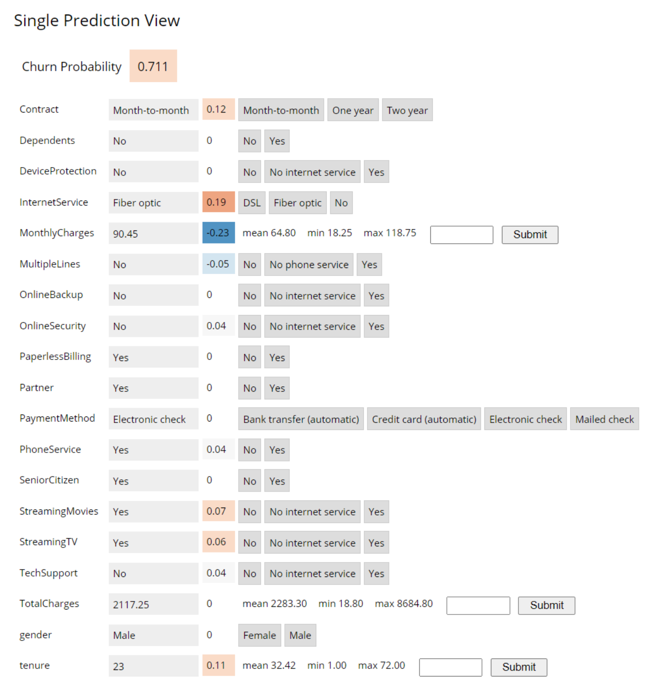
Refractor

id	Probability	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	Month
1241	0.711	Male	Yes	Yes	No	23	Yes	No	Fiber	No	No	No	No	Yes	Yes	Month	Yes	Elect	90.45
1300	0.703	Male	Yes	Yes	No	15	Yes	Yes	Fiber	No	No	No	No	No	Yes	Month	Yes	Bank	85.6
5751	0.625	Male	Yes	Yes	No	16	Yes	No	Fiber	No	No	Yes	Yes	No	Yes	Month	Yes	Elect	91.55
6960	0.619	Femal	Yes	Yes	No	18	Yes	Yes	Fiber	No	No	No	Yes	Yes	Yes	Month	No	Bank	99.75
4774	0.600	Male	No	No	No	5	Yes	No	Fiber	No	No	No	No	No	Yes	Month	No	Maile	78.75
4470	0.543	Femal	No	No	No	3	Yes	No	DSL	No	No	No	No	Yes	No	Month	Yes	Elect	54.2
3074	0.504	Femal	No	No	No	4	Yes	No	DSL	No	Yes	No	No	Yes	Yes	Month	Yes	Bank	68.65
6440	0.217	Male	No	No	No	1	Yes	No	No	No in	No in	No in	No in	No in	No in	Month	No	Maile	19.55
2893	0.150	Femal	No	No	No	6	No	No ph	DSL	Yes	Yes	Yes	Yes	Yes	Yes	Two y	Yes	Maile	63.4
1549	0.059	Male	No	No	No	26	Yes	No	No	No in	No in	No in	No in	No in	No in	Month	No	Maile	20.9

Figure 14: A CML generated application for prediction diagnostics

- The Probability column (i.e. the second column) in the table in Figure 14 represents the predicted value for the target columns, with the other columns representing feature values for the prediction. Using underlying explainability code, the columns with the most influence on the predicted value, for a particular prediction, are color coded. Here, cells with a darker red color represent those with a positive effect on the predicted value, while blue cells represent those with a negative effect. Cells with no color shading represent those with little impact in either direction.

- For this sample application, built using the Python-based Flask Web framework (CML can also host applications built using the R language-based Shiny framework), any of the individual rows can be clicked on to display detail for just that particular prediction, as shown in Figure 15. Here, the predicted value is shown at the top of the screen and each feature column is displayed on a separate row. For each such column, its name, value, and the color-coded positive or negative impact on the prediction, is shown. To the right of the color-coded impact figure is the list of all possible values for the column. Clicking on any of those values allows a what-if analysis to take place, where the Application interacts with the deployed Model's rest endpoint to get the new predicted value. The alternate predicted value and influence figures for each column, will be displayed interactively.


Single Prediction View

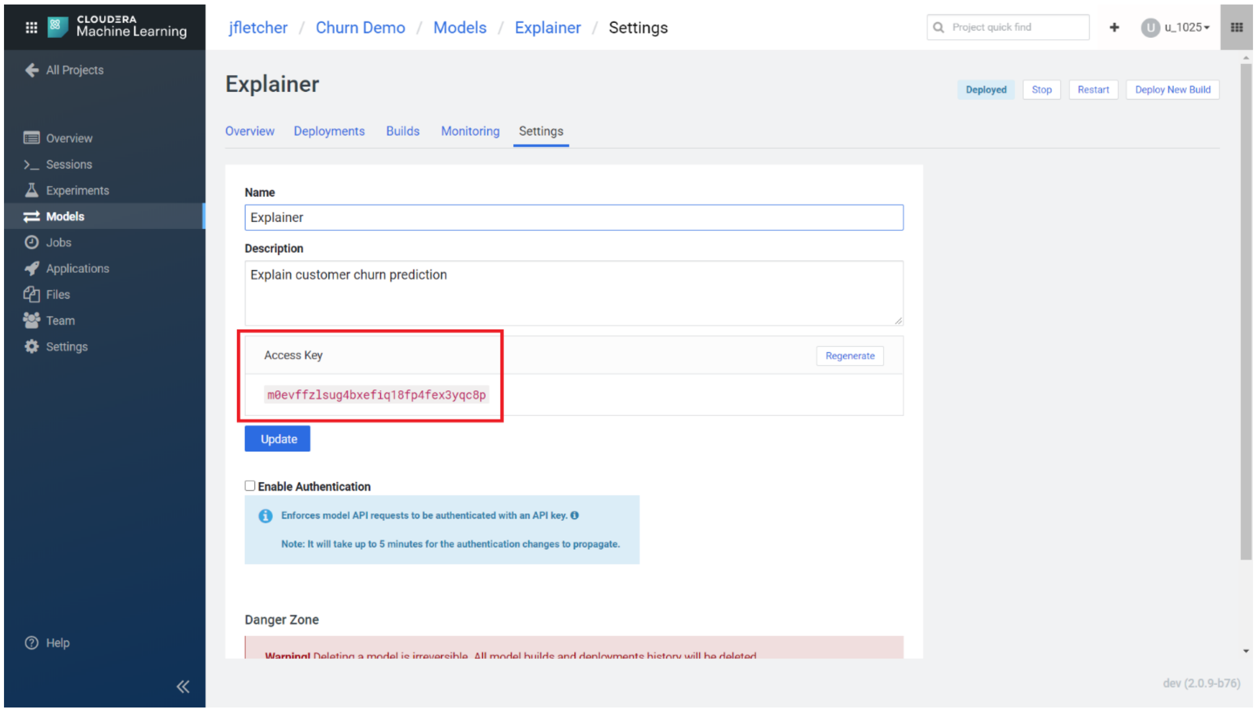
Churn Probability 0.711

Contract	Month-to-month	0.12	Month-to-month	One year	Two year		
Dependents	No	0	No	Yes			
DeviceProtection	No	0	No	No internet service	Yes		
InternetService	Fiber optic	0.19	DSL	Fiber optic	No		
MonthlyCharges	90.45	-0.23	mean 64.80	min 18.25	max 118.75		Submit
MultipleLines	No	-0.05	No	No phone service	Yes		
OnlineBackup	No	0	No	No internet service	Yes		
OnlineSecurity	No	0.04	No	No internet service	Yes		
PaperlessBilling	Yes	0	No	Yes			
Partner	Yes	0	No	Yes			
PaymentMethod	Electronic check	0	Bank transfer (automatic)	Credit card (automatic)	Electronic check	Mailed check	
PhoneService	Yes	0.04	No	Yes			
SeniorCitizen	Yes	0	No	Yes			
StreamingMovies	Yes	0.07	No	No internet service	Yes		
StreamingTV	Yes	0.06	No	No internet service	Yes		
TechSupport	No	0.04	No	No internet service	Yes		
TotalCharges	2117.25	0	mean 2283.30	min 18.80	max 8684.80		Submit
gender	Male	0	Female	Male			
tenure	23	0.11	mean 32.42	min 1.00	max 72.00		Submit

Figure 15: CML generated app, in single prediction view

This application is hosted by CML and the value here is extremely high. The application at once provides a test harness for the model deployment and facilitates a much deeper and interactive explainability analysis than would be possible by running predictions and rendering their explainability visualizations one at a time.

Note that in order for this code to work in the case of models deployed with authentication, the CML user must obtain the model's access key, as highlighted in Figure 16, and insert it into the application's code, which must then be saved, as shown in Figure 17.



Name: Explainer

Description: Explain customer churn prediction

Access Key: m0evffzlsug4bxefiq18fp4fex3yqc8p

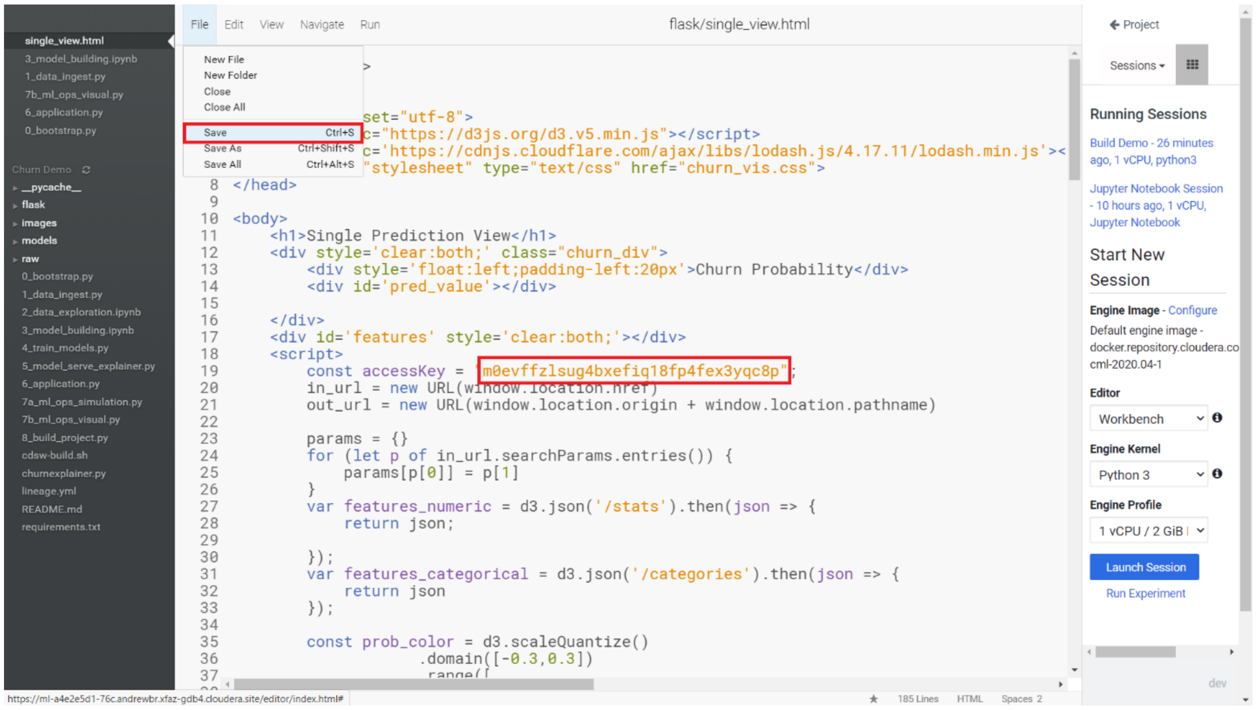
Enable Authentication

Enforces model API requests to be authenticated with an API key.

Note: It will take up to 5 minutes for the authentication changes to propagate.

Warning! Deleting a model is irreversible. All model builds and deployments history will be deleted.

Figure 16: The model's access key is displayed, and can be copied from, the Settings tab of the Model screen



```
>

set="utf-8">
c="https://d3js.org/d3.v5.min.js"></script>
c='https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/lodash.min.js'><
"stylesheet" type="text/css" href="churn_vis.css">
</head>

<body>
    <h1>Single Prediction View</h1>
    <div style='clear:both;' class="churn_div">
        <div style='float:left;padding-left:20px'>Churn Probability</div>
        <div id='pred_value'></div>

    </div>
    <div id='features' style='clear:both;'></div>
    <script>
        const accessKey = "m0evffzlsug4bxefiq18fp4fex3yqc8p";
        in_url = new URL(window.location.href)
        out_url = new URL(window.location.origin + window.location.pathname)

        params = {}
        for (let p of in_url.searchParams.entries()) {
            params[p[0]] = p[1]
        }
        var features_numeric = d3.json('/stats').then(json => {
            return json;

        });
        var features_categorical = d3.json('/categories').then(json => {
            return json
        });

        const prob_color = d3.scaleQuantize()
                    .domain([-0.3,0.3])
```

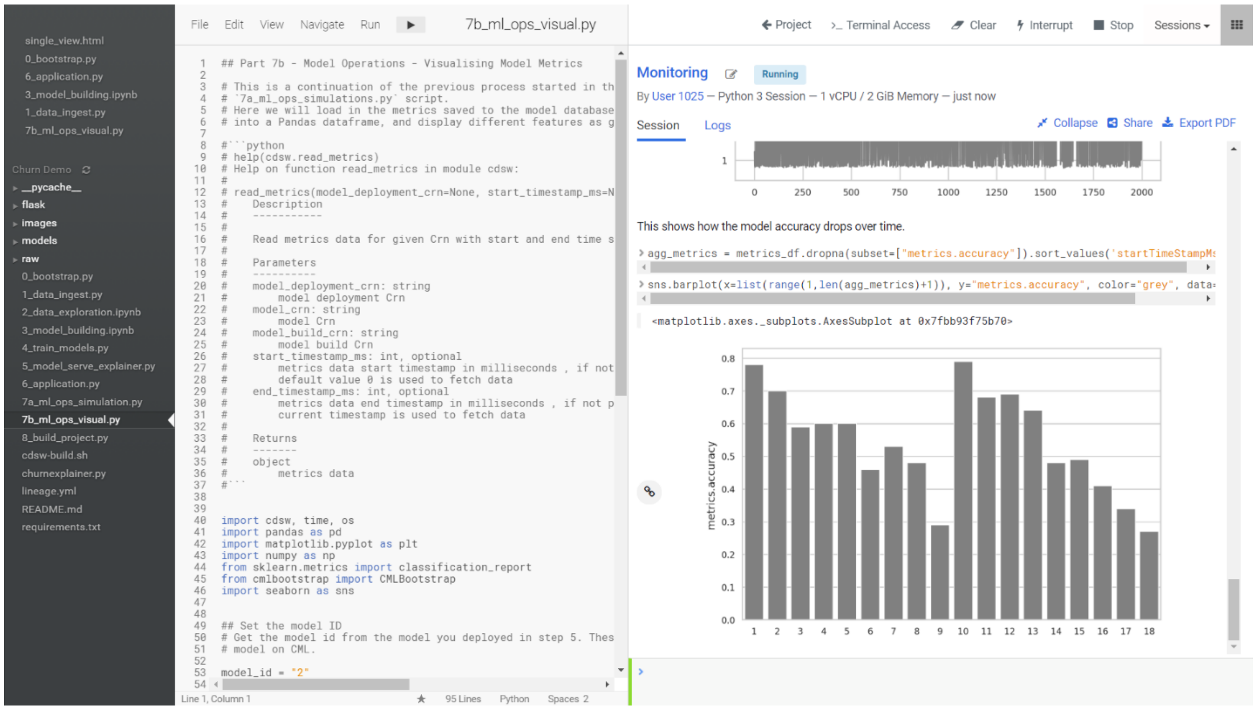
Figure 17: Inserting the model's access key into the sample application's code

MONITORING

The final step in the machine learning workflow is one that all too often has been an afterthought for adopters and vendors in the AI space: monitoring model performance, accuracy and drift. Again, the layered approach of concrete structures at CML's base, with open source components and free-form visualizations from code, allows monitoring to be included in the ML workflow by convention.

But it gets even more sophisticated: because of the way that CML logs and catalogs models, builds, deployments and predictions, its API has the ability to judge the accuracy of predictions well after they're made, through the integration of so-called "ground truth." Essentially, using the API's read_metrics function to recall historical predictions and, optionally, the track_delayed_metrics function to register and update with the actual outcomes, the two can be compared to judge the accuracy and efficacy of models after the fact.

In addition to prediction-by-prediction accuracy tracking, it's also possible to track aggregate accuracy using CML's track_aggregate_metrics API function. This can be done at a predetermined granularity, to monitor drift, and all of this can be run on a scheduled basis – once per day, for example – using CML's Jobs facility. The periodic comparisons can then be visualized in classification reports, as shown in Figure 18.



```
## Part 7b - Model Operations - Visualising Model Metrics

# This is a continuation of the previous process started in th
# `7a_ml_ops_simulations.py` script.
# Here we will load in the metrics saved to the model database
# into a Pandas dataframe, and display different features as g

#```python
# help(cdsw.read_metrics)
# Help on function read_metrics in module cdsw:
#
# read_metrics(model_deployment_crn=None, start_timestamp_ms=N
#    Description
#    -----------
#
#    Read metrics data for given Crn with start and end time s
#
#    Parameters
#    ----------
#    model_deployment_crn: string
#        model deployment Crn
#    model_crn: string
#        model Crn
#    model_build_crn: string
#        model build Crn
#    start_timestamp_ms: int, optional
#        metrics data start timestamp in milliseconds , if not
#        default value 0 is used to fetch data
#    end_timestamp_ms: int, optional
#        metrics data end timestamp in milliseconds , if not p
#        current timestamp is used to fetch data
#
#    Returns
#    -------
#    object
#        metrics data
#```


import cdsw, time, os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report
from cmlbootstrap import CMLBootstrap
import seaborn as sns


## Set the model ID
# Get the model id from the model you deployed in step 5. Thes
# model on CML.

model_id = "2"
```

This shows how the model accuracy drops over time.

```
> agg_metrics = metrics_df.dropna(subset=["metrics.accuracy"]).sort_values('startTimeStampMs
> sns.barplot(x=list(range(1,len(agg_metrics)+1)), y="metrics.accuracy", color="grey", data=
<matplotlib.axes._subplots.AxesSubplot at 0x7fbb93f75b70>
```

Figure 18: Monitoring model accuracy

Explicit support for ground truth tracking and integration of that data into the monitoring process is a powerful capability set. It enhances the sophistication of model management and the thoroughness of accuracy monitoring. Along with its support for model explainability analysis, CML greatly enhances the customer's ability to practice responsible AI more proactively. This has a positive cascading impact on customer trust, risk management and successful digital transformation.

GOVERNANCE

In addition to the individual steps in the ML development and operational management processes, tracking the big picture of a model's progress through the development and production workflow is important as well. Just as with tracking the lineage of data in conventional analytics, tracking the lineage of machine learning models in predictive analytics is critical.

Cloudera, as the key backer behind the open source Apache Atlas data governance platform, has extended that technology to track ML assets as well as data sets. Furthermore, it has integrated that capability with CML, using the CRNs and UUIDs for models, builds, deployments and predictions to track ML lineage, all the way from data ingest to final prediction. This is shown in Figure 19.

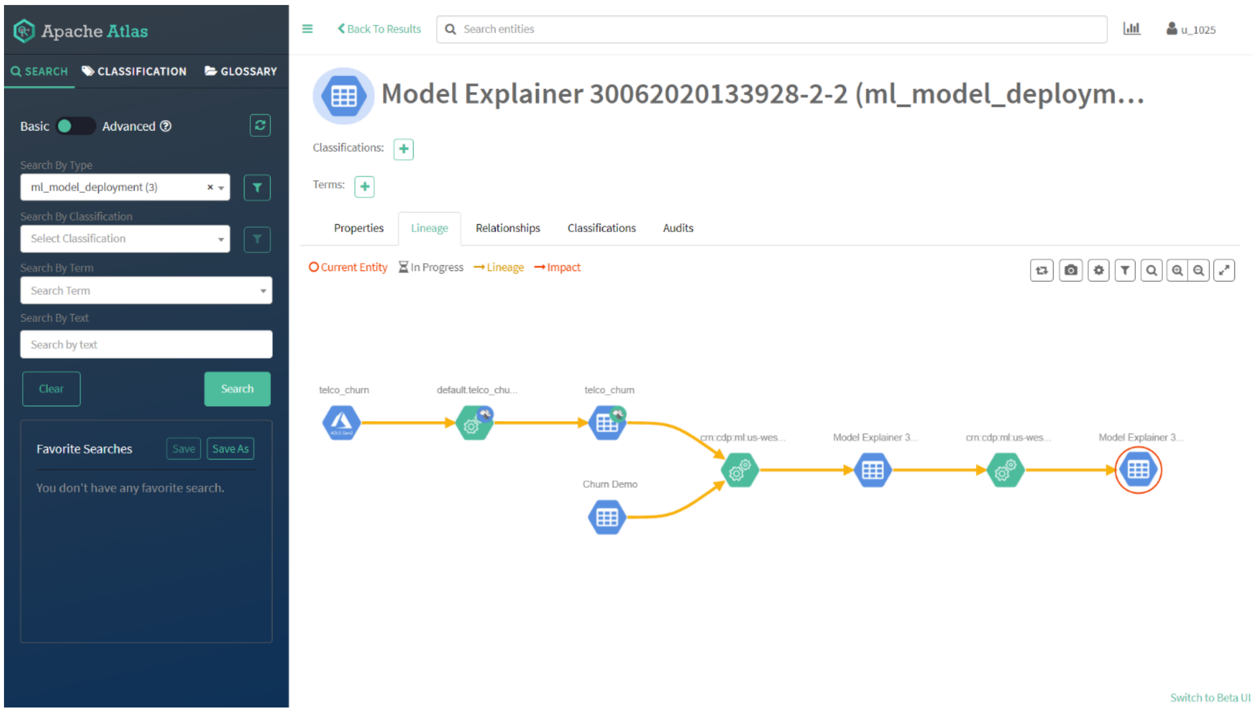


Figure 19: CML facilitates model lineage in Apache Atlas

This lineage graph shows how the dataset at the far left, which originates in cloud storage (Azure Blob Storage in this case), proceeds to go through ingest and transformation, is combined with test data to produce a model, and is then used in inferencing to predict a churn probability. This lineage visualization is the same type used for data sets in conventional analytics; Apache Atlas handles it all, and Cloudera Machine Learning ties it together with the development and management of the ML assets themselves.

CONCLUSION

Even if machine learning development has emerged as a segregated and highly-specialized pursuit, it is in fact a sub-discipline of software development. As such, ML work needs to be pursued with the same engineering rigor around testing, deployment, operational monitoring and collaborative development that mainstream software work is. If all of this makes machine learning sound less "special," so be it, as this will allow the technology to become truly pervasive throughout enterprise organizations, just as application development is today.

"Citizen data scientists" have automated machine learning at their disposal. But formal data scientists and machine learning engineers need their own tool chain in order to make AI in the enterprise, at scale, a reality. Furthermore, the ethical requirements around ML, in terms of model accuracy, explainability and mitigation of bias, greatly benefit from tooling support as well.

Cloudera Machine Learning supplies all of this support, in an environment that can be as prescriptive or permissive as the customer may require. CML provides all the tooling necessary for productive data science work, and yet integrates with open source frameworks and development tools to accommodate organizations which may have standardized on a certain subset of them. Its layering of formal constructs, conventions and extensibility provides the combination of structure and flexibility necessary for the current state of data science, which is transitioning from a preponderance of ad hoc, craftsperson-like approaches to a more standardized, disciplined approach.

CML's capabilities are based on customer requirements and demands. It will continue to evolve along those lines, even as it introduces best practices to its customers and mentors them into scalable practices and techniques. Hopefully, this report has provided helpful insight not only into the mechanics and capabilities of CML, but also the motivation behind them, and the significant industrial/business advantages they deliver.