Kubeflow

A comparison of Kubeflow & TFX

SpringML

www.springml.com

Introduction

In this report, we compare two technologies that have come out of Google for managing machine learning pipelines.

The first is **Kubeflow**, which has been in development since 2018 and was originated as a way of bringing the ideas of TFX (used only internally at Google at the time) to the public via open source tools and is in the process of changing as many developments as open source tools come and go.

The second is **TensorFlow Extended (TFX)** itself. Google announced that it would be making TFX available to the public at the end of 2018.

Kubeflow

The main mission of Kubeflow is to make scaling machine learning (ML) models and deploying them to production as simple as possible across production environments, by letting Kubernetes do what it's great at:

- Easy, repeatable, portable deployments on a diverse infrastructure (laptop <-> ML rig <-> training cluster <-> production cluster)
- Deploying and managing loosely-coupled microservices
- Scaling based on demand

At a high level, the execution of a pipeline proceeds as follows:

Python SDK :
You create components or specify a pipeline using the Kubeflow Pipelines domain-specific language (DSL)

DSL compiler :
The DSL compiler transforms your pipeline's Python code into a static configuration (YAML)

Pipeline Service :
You call the Pipeline Service to create a pipeline run from the static configuration

Kubernetes resources : The Pipeline Service calls the Kubernetes API server to create the necessary Kubernetes resources (CRDs) to run the pipeline

Orchestration controllers : A set of orchestration controllers execute the containers needed to complete the pipeline execution specified by the Kubernetes resources (CRDs). The containers execute within Kubernetes Pods on virtual machines. An example controller is the Argo Workflow controller, which orchestrates task-driven workflows

Artifact storage: The Pods store two kinds of data:
Metadata: Experiments, jobs, runs, etc. Also single scalar metrics, generally aggregated for the purposes of sorting and filtering. Kubeflow Pipelines stores the metadata in a MySQL database.
Artifacts: Pipeline packages, views, etc. Also large-scale metrics like time series, usually used for investigating an individual run's performance and for debugging. Kubeflow Pipelines stores the artifacts in an artifact store like Minio server or Cloud Storage

Persistence agent and ML metadata: The Pipeline Persistence Agent watches the Kubernetes resources created by the Pipeline Service and persists the state of these resources in the ML Metadata Service. The Pipeline Persistence Agent records the set of containers that executed as well as their inputs and outputs. The input/output consists of either container parameters or data artifact URIs

The MySQL database and the Minio server are both backed by the Kubernetes PersistentVolume (PV) subsystem

Pipeline web server: The Pipeline web server gathers data from various services to display relevant views: the list of pipelines currently running, the history of pipeline execution, the list of data artifacts, debugging information about individual pipeline runs, execution status about individual pipeline runs

Deploying Kubeflow Pipelines

The steps to deploy Kubeflow Pipelines are enumerated in our “Deploying Kubeflow Pipelines” document.

What are some pros and cons of Kubeflow?

Pros

Kubeflow makes it very easy for data scientists to get distributed training and serving on a cluster without worrying about infrastructure

A variety of open-source tools have been combined to bring together several good ideas

Kubeflow introduced the concept of fairing which makes it easier for data scientists to implement jobs directly in jupyter notebooks along with model implementations

Kubeflow supports multiple frameworks such as Tensorflow, Pytorch, MXNet, etc.

Cons

Ksonnet which is the prime component for configuration on Kubeflow will be discontinued in further versions, and the community will need to replace it. This will cause some major changes in Kubeflow code

There are several issues with sample pipelines and notebooks present on kubeflow website. These issues are still getting fixed

The variety of options of open-source tools means the user must be aware and informed about the tools available and make informed decisions about which components to use in any given deployment, which leads to a less clearly defined, universal pipeline

The variety of tools also means the user needs to learn multiple technologies with different languages to utilize all of their desired tools within Kubeflow

TensorFlow Extended (TFX)

TensorFlow Extended is Google’s internal pipeline for managing the full Machine Learning process from raw data ingest through **training, evaluation, and deployment** across multiple platforms, now made available to the public.

In a nutshell, TFX allows teams to:

- **Divide work on a machine learning pipeline** into discrete modules with well-defined interfaces for interacting with one another
- At the same time, **keep all high-level instructions** for the model in one place for transparency throughout the process
- **Approach machine learning model development not as a "one and done" process, but as a process** like other software development that considers iteration and improvement part of its core
- Makes it (relatively) **easy to ingest and build from Google** and other organization's trained models
- Makes it (relatively) **easy to deploy models** to the web (server or client browser), edge devices, contribute them back to the community, etc.

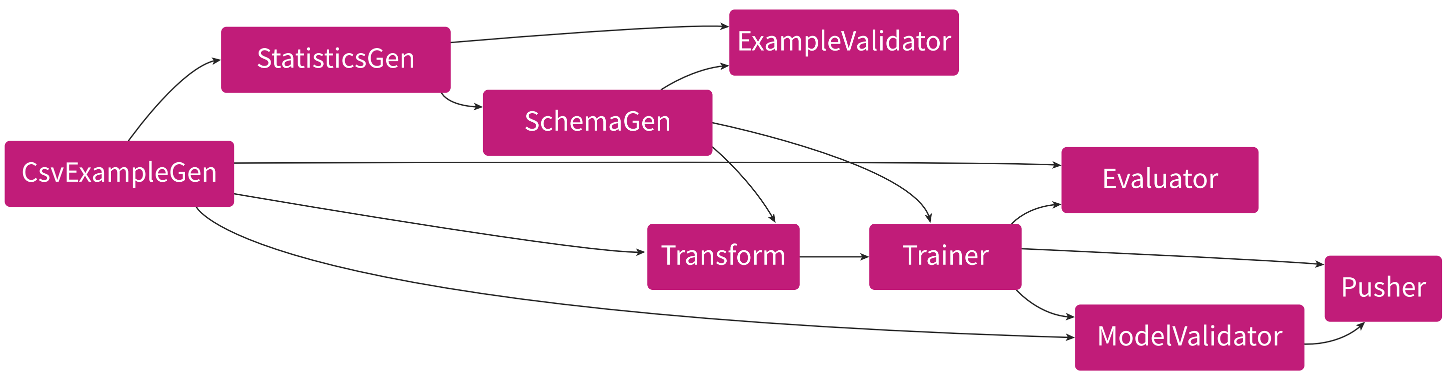
?

What runs the pipeline?

The pipeline can be run by Airflow or Kubeflow Pipelines. Airflow seems to be more prominent positioned in Google's messaging, but that may be because of current adoption in the field.

What components make up the pipeline?

The pipeline is made up of a series of 'modules' that each do a specific task. In addition to allowing customizability where you need it, this helps reduce the cognitive load of considering the entire pipeline at once, and lets Data Engineering teams focus on one aspect of the pipeline at a time (or provides for conveniently clear boundaries for distributing work on various aspects of the pipeline across engineers).



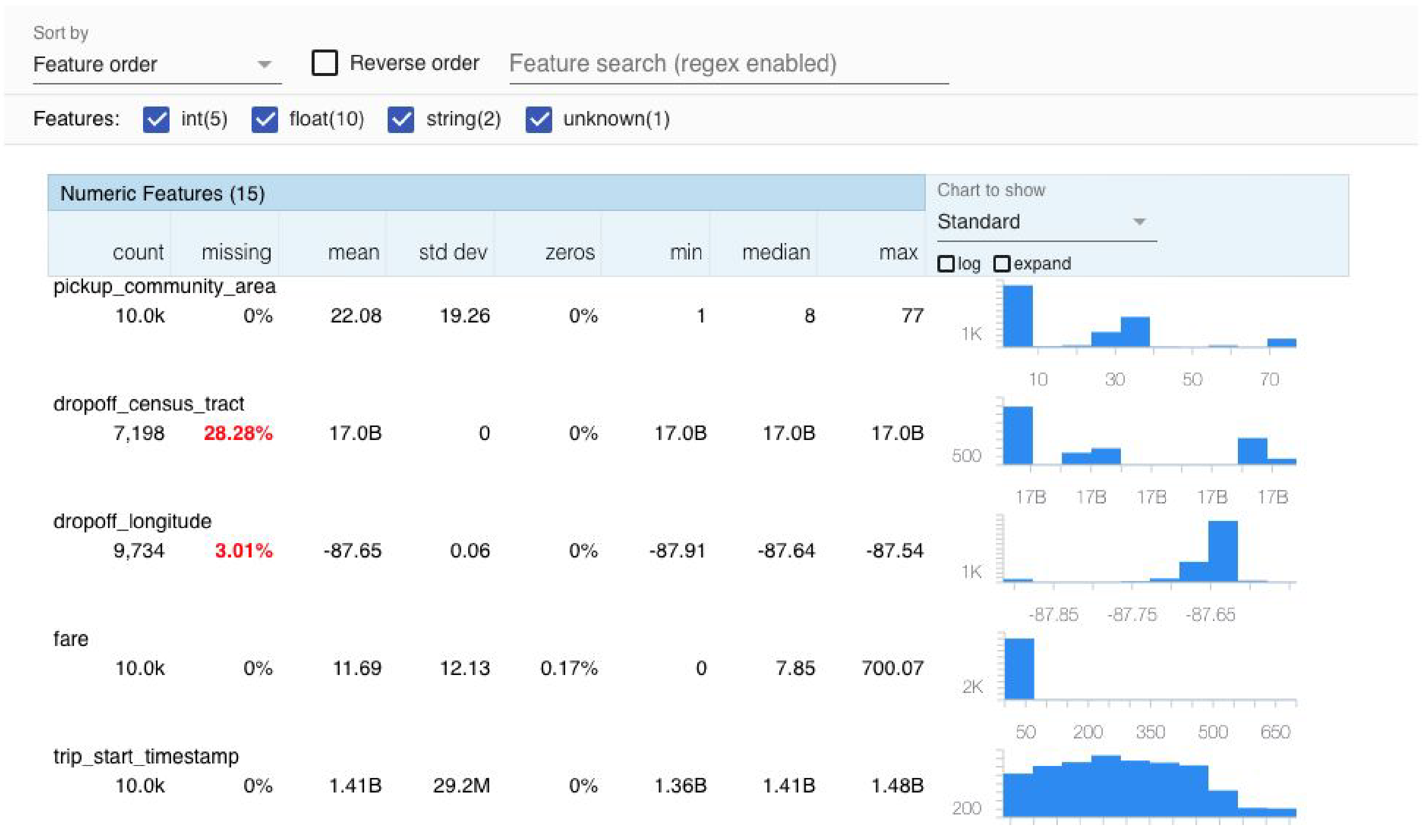
The above diagram shows how the Pipeline appears within an Airflow job.

Those modules are:

CsvExampleGen	which ingests data from raw sources like CSVs, BigQuery, TFRecords, splits that data into training and evaluation sets
StatisticsGen	uses TensorFlow Data Validation to provide a window into the raw dataset, calculating descriptive statistics to identify missing data, values outside expected ranges, etc. This view also allows a data scientist to examine the features and check for other data quality problems like post-treatment variables
SchemaGen	is one of the simpler components of the pipeline. It examines the statistics created from StatisticsGen and creates a data schema to represent the data types of each feature column, and whether they are required fields or not. It creates the schema automatically, but developers are expected to review and modify the schema to confirm its accuracy
ExampleValidator	looks for anomalies and missing values in the dataset. For example, it can detect training-serving skew and data drift as it compares training data to evaluation data or new data coming in during serving

Transform	performs consistent feature engineering on all subsets of the dataset (i.e. train, dev, test). Here we engineer our features, vocabularies, embeddings, etc and can define features as categorical, dense, and bucket features for use in different kinds of complex models. The same Transform module can apply to both training data and incoming data in production in order to ensure that the transformations are consistent across both datasets
Trainer	trains the model using TensorFlow Estimators
Evaluator	performs deep analysis of the training results. In addition to standard model evaluation statistics, it can show statistics on user-defined slices of the data. This can help test whether your model is performing too poorly on a particular subgroup of input data, as well as providing insight into what additional data you need to gather or additional features you need to engineer in order to improve. Evaluator also helps trackperformance over time to see how model iteration is improving performance. This module is based on TF Model Analysis
ModelValidator	is like an automated gatekeeper that ensures that new versions of the model are "good enough" to be pushed to production, especially when new models are trained and served automatically. It takes two models: the last good evaluated model, and the current model coming from training. It "blesses" the new model if performance is the same or better. Fails if not 'better' than the last one. It also does this evaluation on new data, which helps make sure the model is ready for production data rather than simply overfitting the training data
Pusher	deploys the model to serving infrastructure, like TensorFlow.js, TensorFlow Serving and TF Hub

Example code to deploy a TFX pipeline to Airflow can be found in this taxi_pipeline.py example. That pipeline is supported by this taxi_utils.py library. Those files come from Google's official TFX Developer Tutorial.



Numeric Features (15)

	count	missing	mean	std dev	zeros	min	median	max
pickup_community_area	10.0k	0%	22.08	19.26	0%	1	8	77
dropoff_census_tract	7,198	28.28%	17.0B	0	0%	17.0B	17.0B	17.0B
dropoff_longitude	9,734	3.01%	-87.65	0.06	0%	-87.91	-87.64	-87.54
fare	10.0k	0%	11.69	12.13	0.17%	0	7.85	700.07
trip_start_timestamp	10.0k	0%	1.41B	29.2M	0%	1.36B	1.41B	1.48B

StatisticsGen Output: This is an example of the metadata you can see on each feature output by the StatisticsGen module.

That all sounds very cool.
What are some other pros and cons of TFX?

Pros

Organization & Version Control - The biggest pro is the organization that it brings to the pipeline with a clear, version-controllable approach to the whole machine learning process

Focus with Integration - It also allows developers to focus on one aspect of the pipeline at a time (or different developers can own different elements) in an organized, coherent fashion

Data Lineage - It provides data lineage features, so you always know what transformations have been performed on your data by the time of model training or other stages in the pipeline. Data lineage, FTW!

Consistent Transforms at Different Stages with Powerful Feature Types - The Transform capabili ties are very empowering, and the template's demonstrated use of _CATEGORICAL_FEA TURE_KEYS, _DENSE_FLOAT_FEATURE_KEYS, and _BUCKET_FEATURE_KEYS allow data scien tists/engineers to easily identify feature types and build some really sophisticated models without even realizing it. Concretely, TensorFlow's DNN LinearCombinedClassifier can treat different features differently in order to build a model that passes appropriate features (like continuous numerical features) through a neural network while letting other features (like categorical dummy variables) bypass that network

NLP Feature-Engineering Tools - Great NLP vocab tools like multiple buckets into which to hash out-of-vocab words, easy definitions of vocab size

Versatile SavedModel Format + Separate Eval SavedModel - The pipeline provides for separate SavedModel for production and and EvalSaved Model for evaluation, which can help make production models more lightweight and allow for evaluation on less expensive hardware than TPUs used for training, for example

Cons

Challenging for Quick, Individual One-Off Projects - This is a complex system that an organi-zation would need to really adopt at minimum on an entire project level

Learning Curve Does Not Start at Ground Level - While we believe discretizing the pipeline into different modules actually makes things easier to understand, this is still starting at "Floor 3" rather than the "Ground Floor" and thus needs teams able to understand and work with all the compo-nents. You're not just building a simple test example here, but an entire production pipeline

New Software - It is still new, so no doubt we will see some changes over the next year

Underlying Architecture in Development - It uses Apache Beam, which isn't fully released for Python 3 yet (although it looks like we are close), so there may be some environment reconfigura tions required as it develops

Supports Only TensorFlow Framework - This seems to currently only be applicable to TensorFlow, so models built in PyTorch, Caffe, MXNet, etc. may not be able to fit in well with an organization that adopts this methodology

Deployment Methodology

We do not recommend teams start with a blank file when coding a TFX pipeline. There are lots of interdepecncies and calls to the metadata store as each module looks up the details it needs to interact with the model. We recommend always beginning with one of Google's example pipelines, and modifying it for the current model as needed.

Thoughts on TFX

TFX seems very powerful for machine learning teams.

One metaphor that the Google team has used to describe TensorFlow Hub, which is part of this new TensorFlow ecosystem, is that these tools are adding to the machine learning community what adding version control was for software engineering. That may be more literally true of TensorFlow Hub, but the concept still applies here. This turns a human machine learning engineer's actions into code and allows the entire pipeline to be documented and checked into version control.

A Comparison Between Kubeflow and TFX

	Kubeflow	TFX
Ease of Use / Documentation	Documentation is up-to date. However, Kubeflow samples are still not running properly and there are open issues on github	A steep learning curve to understand the full pipeline, but it is straightforward once you do. Documentation is up-to-date and templates to work from are provided, but full documentation on the modules is still being added, so seeing all your options requires some digging
Underlying Philosophy & Architecture	A suite of open-source tools from different teams that come packaged to run machine learning pipelines on a Kubernetes cluster. It includes a pipeline server (Kubeflow Pipelines) and metadata storage databases	Google's internal tools for managing ML pipelines are now being released publically. They run on an Apache Beam Pipeline with a supporting SQL MetadataStore database and are deployed by Airflow or Kubeflow Pipelines

Data cleanup / Ingest	Nuclio functions, Minio	StatisticsGen module uses TensorFlow Data Validation
Data lineage / tracking	Nuclio functions	MetadataStore provides data lineage information to clarify what transforms have been applied to data
Model training	Training using modules MPINet, MXNet , PyTorch, Tensorflow Training	The Trainer module makes heavy use of TensorFlow API for training
Hyperparameter tuning	Available via the Katib component	Still in development. Possible to call AI Platform (by setting the Trainer's `executor_class` to an AI Platform Executor), but plumbing to connect that to the Trainer module sounds like it is still in the works.
Distributed training	TFJob CRD in Kubernates cluster. Configuration of nodes can be done by ksonnet	Still in development. Possible to call AI Platform (by setting the Trainer's `executor_class` to an AI Platform Executor), but plumbing to connect that to the Trainer module sounds like it is still in the works.
Iterative Training	Supports iterative training by different docker versions using ksonnet	Purpose-built to facilitate iterative training and automatic deployment (with automatic tests to ensure new models are improving)
Model Output	Model output saved	Outputs SavedModel format which can be further trained or deployed to production
Model Serving (For Inference at Scale)	Serves in containers via Seldon	Pushes to TF Serving for serving in containers as well as TF.js, AI Platform & TF Hub

Accelerated Hardware	GPU Support (Supports TensorRT). Chainer Training component can be used for CUDA operations	TF Serving supports GPU-acceleration by converting model to TensorRT
Version control of infrastructure & code	Nuclio functions, version control using ksonnet. Versions of models can be managed using ModelDB	Entire infrastructure is code and able to be checked into version control
Monitoring UI	Argo UI , TFJobs Dashboard, Katib UI (Requires Ambassador)	Monitoring of training via TensorBoard & Pipeline status via Airflow / Kubeflow Pipelines UI
Model Performance Monitoring	Tensorboard, Grafana dashboard, Istio integration	Training is monitored via TensorBoard & comparison between model versions is done via the Evaluator module (using TF Model Analysis)
Platforms supported	Tensorflow, PyTorch, XGBoost	TensorFlow, Keras (with TensorFlow backend) in the works
Future Outlook	A core module, ksonnet, is being replaced, so we expect significant changes	Minor changes likely due to its newness

Model Operationalization

Outlines things to consider when putting a model pipeline in production.

How do you know when it's time to retrain your model?

The right time to retrain your model will naturally depend upon your specific business needs, but there are a few signs that it might be time:

YOU HAVE ACCESS TO MORE DATA

More data has often been shown to have a stronger impact on model performance than the selection of a more advanced algorithm. If you have significantly more data from your model's initial deployment, and you can label that data (if your model requires it), retraining may lead to a significant boost. This is another good reason to save your training curves from all training runs, as examining those could help a data scientist tell you whether more data will help in your case.

YOU HAVE ACCESS TO MORE DIVERSE DATA THAT IS MORE REPRESENTATIVE OF THE BREADTH OF DATA ON WHICH YOU WANT TO BE ABLE TO PREDICT

If you're getting new data from users, it is very likely more diverse than the original data on which you trained your model. Taking a computer vision mode as an example, new data may include more lighting conditions, more races and ages of people, and many more device types capturing the images. Retraining on more diverse data that represents the kind of data on which you would like to be able predict can help your model perform better in those real-world cases.

THE VOCABULARY OF THE DATA SOURCE HAS CHANGED OVER TIME

Language changes all the time, and slang changes even more quickly, so a Natural Language Processing model focusing on a colloquial datasource like Twitter, online reviews, or even news sources needs to be updated with some regularity. Even in more industrial settings, new terms are introduced as new technologies are developed and new products are introduced. If your model is based on data that has some changing characteristics or vocabulary, choosing the right interval at which to retrain your model will help you keep up with what the kids are really saying or how those new hyperwidgets are being described in maintenance reports

YOUR DESIRED LEVEL OF GRANULARITY HAS CHANGED

Many features are binned before being fed into a model, and if you need to bin your numerical features in a new way, such as turning quarterly data into monthly data for more specific seasonality predictions, this might also prompt a retraining of your model. This doesn't mean that you should always start with the most fine-grained bins. Often, if you didn't have enough data upon initially training your model, binning can help improve accuracy. But as you collect more data, you can make more fine-grained predictions.

YOU'D LIKE TO ADD ADDITIONAL INPUTS OR OUTPUTS

If you'd like to add the ability to identify a new class or take in a new feature for your market predictions, you'll need to retrain.

?

When should a model be promoted to production as the "live" version?
What criteriashould be applied to determine that a new model is now production ready?
How does TFX compare with Kubeflow in this regard?

When training a new version of a model, it should always need to "pass inspection" before being pushed to production. Additionally, keeping track of how your model is improving across different versions is also important. Here's how the systems compare in how they approach this:

Kubeflow

Kubeflow pipelines have an excellent feature to compare accuracy metrics of several runs such as roc-auc-score, accuracy score, etc. With this regards, one of the best components included with Kubeflow is Seldon. Seldon allows us to compare two models through A/B testing as well as well as allow. Best model based on results push for serving.

TFX

TensorFlow Extended has two modules dedicated to this concept of model evaluation.

- **Evaluator**
  is designed to provide graphs and statistics for humans to interpret the model. As mentioned in the TFX module descriptions above, it uses TF Model Analysis to allow you to dive into your model performance to see on which subsets of your data it is performing well, and where it can be improved. This would be where a data scientist or engineer evaluating a model (or a series of versions of models) would focus.
- **ModelValidator**
  Which is more of an automated gatekeeper. The ModelValidator compares each newly trained model to the last approved model and only pushes it to production if it is equal or better according to the metric you are tracking. This is what would be used by an automated system that pushes new models out if they are being retrained regularly.

? How does model inference scale in production, especially for real time predictions via API?

Both Kubeflow and TFX produce models that are ready to be hosted in a number of deployment scenarios. For most online use cases, that will mean serving a model as a pod in a Kubernetes cluster which can be autoscale to support increased traffic.

Kubeflow initially supported the containerization of models through a component called Seldon. It now supports deploying models to containers through Seldon and TensorFlow Serving (TF Serving), Google's newer model-containerization system. TF Serving models can be accessed via REST or gRPC calls.

TensorFlow Extended also outputs models that are easily deployed to TF Serving containers. It can also deploy models to TensorFlow.js to serve via Javascript in a browser. TensorFlow.js works surprisingly well on even computer vision models using the webcam and would be a very fast option for many models served online.

Tensorflow Serving pods can be deployed via Kubernetes so that autoscaling kicks in as CPU utilization creeps up. Both pipelines also support running containerized models on the GPU as well for hardware-accelerated predictions, at which point a custom metric can be defined to trigger autoscaling based on GPU utilization.

Final Recommendations

We think both of these technologies are very interesting, and that as they continue to develop, it will become easier for teams to incorporate machine learning into even more aspects of their businesses.

That being said, some critical development is currently in progress for both frameworks in the months to come (Kubeflow replacing ksonnet with Kustomize and TFX getting full support for Python 3 on Apache Beam and reaching the official launch of TensorFlow 2.0). We do feel like both platforms are ready for more experimental R&D projects, however, and that learning one or both would help teams be prepared for the stable launches to come.

We think some exceptions where we would deploy one or the other (or both together) right now are the following:

- We would use Kubeflow if we needed to train a large, distributed model across many machines, particularly if it were on-premises
- We would use Kubeflow if we were considering an infrastructure change and wanted to make sure our pipelines were well-prepared to run on another cloud provider or hardware deployment
- We would use TFX if our system were designed to automatically retrain and deploy a new model at regular intervals, such as nightly, as it learns from additional user data
- We would use TFX if we trained many versions of similar models within our organization (for example many topic clustering models focusing on different domains or types of text input) and wanted a smooth, repeatable process for training

Both technologies are interesting, and we expect to see much more development and maturation of this machine learning pipeline space in the years to come.