ML Pipelines

← Tensorflow Extended
Apache Beam →

•••

Western Greek Software Developers
Meetup # 17 -- global.gotomeeting.com/join/261465869

Kubeflow

About me

- Comp. Engineering Student - Univ of Patras
- Coding > 10 years now
- Deep Learning & Software Engineering

- More at https://ntakour.is

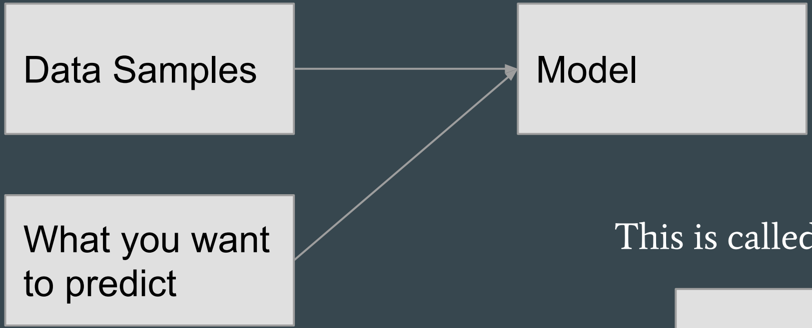
Theodoros Ntakouris

Presentation Index

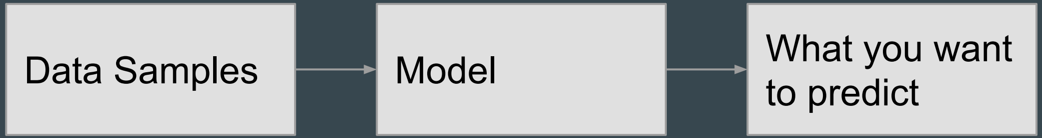
- Introduction
  - What exactly are ML pipelines?
  - Challenges and Problems
  - Quick Framework/Library Evaluation
- Tensorflow Extended
  - How it works (Apache Beam)
  - Why it works that way
  - Why it's awesome
  - Scaling
  - Available Tooling
- Demo

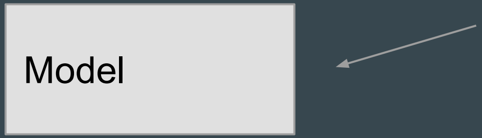
A 1-slide Introduction to Supervised Machine Learning

This is called training



This is called inference





This is usually a mathematical object. It ‘learns’ by predefined rules, given examples and labels for each example.

An easy way to understand ‘learning’ is to calculate if statement thresholds (**if x[‘something’] > [how much]**)

You now know how Supervised Machine Learning works

Introduction -- What are ML Pipelines

Google

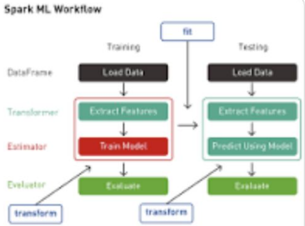
what are ml piplines

All · Images · News · Videos · Maps · More · Settings · Tools

About 87,000,000 results (0.56 seconds)

Showing results for what are ml ***pipelines***
Search instead for what are ml piplines

A machine learning **pipeline** is used to help automate machine learning workflows. ... Machine learning (**ML**) **pipelines** consist of several steps to train a model. Machine learning **pipelines** are iterative as every step is repeated to continuously improve the accuracy of the model and achieve a successful algorithm . Dec 10, 2019



medium.com › analytics-vidhya › what-is-a-pipeline-in-m...
What is a Pipeline in Machine Learning? How to create one ...

About Featured Snippets · Feedback

ML Pipelines -- Common Steps

Input

Preprocessing

Training

Deployment

ML Pipelines -- Common Steps

Input

Reusable, Made to support many data sources

Preprocessing

Configurable for most common preprocessing tasks

Ex. vocabulary building, one hot encoding etc

Training

Can train any model type, or an ensemble, 100% user code

Deployment

Docker ?
Kubernetes ?
AI Platform ?
Model Registry ?

ML Pipelines -- Common PROBLEMS

- Everything has pretty much migrated on the cloud, why don’t we enjoy all the benefits of cloud, too? No need to get a monstrous PC with 4 GPUs in the office to do everything locally.
- Need to scale: **input + preprocessing + training + deployment**

- Code is unit-tested and e2e tested -->
- When writing code that deals with big amounts of data, you need to check data too

ML Pipelines -- Solving *Performance* Problems

- Input
  - Multiple sharded files for parallel reads (similar to HDFS)
- Preprocessing
  - Map Reduce for preprocessing
- Training
  - Multi-GPU
  - Parallel HyperParameter search
- Deployment
  - Multi-Tenant Models
  - Model Compression and Optimization for deployment

Solving Performance Problems: Technical Debt

You want to scale your machine learning model/product/application. You just introduced a ton of technical debt:

- Server Provisioning (Map Reduce, Multi-Worker, Multi-GPU servers on demand)
- Storage Server Provisioning (small, sharded files for parallel reads)
- A ton of new libraries and transactions between systems to do every task fast
- More provisioning for deployments
- Less technical debt if you use the public cloud
- Even less technical debt if your ml pipelines tooling/framework does these for you

ML Pipelines -- Data Problems

- Data change over time and the model should reflect changes in order to be performant (accuracy / precision / recall / top-k error / whatever **metric** you try to maximize)
- Data origin = typically microservices.
- We got to deal with schema changes, distribution changes, trends, null values, unused values, deprecated features/columns, data type changes over time, etc.
- We also got to check if the model performs better against the existing model or some baseline configuration (ex. < 50% accuracy = worse than a random guess)
- And not only just a raw metric, we have to slice over data (acceptable performance on every hour of day for example, not just an overall good score)

More Technical Debt 1/2

- Need more configurable components to check those problems
- Need:
- Statistical distribution checks
- Schema checking
- Input data validation
- Model performance validation
- Infrastructure evaluation (is startup time fast < 5s? Is prediction time fast enough < 200ms ?)
- Alerting infrastructure (slack?)
- Engineer-in-the-loop (inspect visualizations from problems, before each deployment)

More Technical Debt 2/2

- Scale all these up
- Track experiments and performance, training logs over time
- Schedule A/B Tests
- Multi-Version Multi-Labelled Model Deployments (and versioning)

That’s a lot of stuff to make from scratch

ML pipeline library/framework features

- Run “anywhere” and in “any” language
- Each component → Docker Container with Inputs and Outputs (we call those artifacts)
- Some kind of dashboard for experiment tracking and dashboard visualization
- A metadata server (relational database) to track everything
- Alerts, Management UI, Job scheduling included
- Easy to deploy / takes care of orchestration
- Run anywhere = something of (kubernetes, apache spark, apache beam, popular public cloud solutions like GCP Dataflow, Azure Data Bricks, etc)

Popular Solutions

- Kubeflow Pipelines
  - Includes components, orchestration, tracking, dashboards, jobs, visualization ui
  - 3-command deployment on top of kubernetes.
- ML Flow
  - Only runs in Spark
  - Is designed to transition from data science / exploration to production easily
  - Includes a model registry, experiment tracking and serving
- Convrg.io
  - Branding = nvidia certified 'AI OS'
  - Includes experiment tracking, an easy visual pipeline composer
  - Kubernetes Hybrid and Multi-Cloud, bla bla
- TFX

Where are the common components?

- Nowhere.
- Make them yourself
- Or depend on low-popularity < 10 star github repositories, but you still got to wire everything together
- Most still require a docker container as a component

And then, there is Tensorflow Extended (TFX)

- Runs on Apache Beam
  - Meaning it can run on almost every platform, including but not limited to:
  - GCP Dataflow / Azure DataBricks
  - Locally on 1 machine
  - A spark cluster
  - Apache Flink (streaming-processor)
  - Existing Kubeflow Pipelines
- Components Already Provided
  - For 99% of tasks they are sufficient
  - Easy to define custom components with just python code
  - Provided Components scale automatically

What is Apache Beam?

Apache Beam: An advanced unified programming model

Implement batch and streaming data processing jobs that run on any execution engine.

LEARN MORE | TRY BEAM | DOWNLOAD BEAM SDK 2.23.0

JAVA QUICKSTART | PYTHON QUICKSTART | GO QUICKSTART

The latest from the blog

Pattern Matching with Beam SQL

AUG 27, 2020

Improved Annotation Support for the Python SDK

AUG 21, 2020

Performance-Driven Runtime Type Checking for the Python SDK

AUG 21, 2020

How do TFX pipelines look ?

```
return pipeline_def.from_csv(os.path.join(current_dir, 'data')) \
    .generate_statistics() \
    .infer_schema(infer_feature_shape=True) \
    .validate_input_data() \
    .preprocess(user_code_file) \
    .tune(user_code_file,
          train_args=trainer_pb2.TrainArgs(num_steps=5),
          eval_args=trainer_pb2.EvalArgs(num_steps=3)) \
    .train(user_code_file,
           train_args=trainer_pb2.TrainArgs(num_steps=10),
           eval_args=trainer_pb2.EvalArgs(num_steps=5)) \
    .evaluate_model(eval_config=_get_eval_config()) \
    .infra_validate(serving_spec=infra_validator_pb2.ServingSpec(
        tensorflow_serving=infra_validator_pb2.TensorFlowServing(
            tags=['latest']),
        local_docker=infra_validator_pb2.LocalDockerConfig()
    ),
    request_spec=infra_validator_pb2.RequestSpec(
        tensorflow_serving=infra_validator_pb2.TensorFlowServingRequestSpec()
    )) \
    .push_to(relative_push_uri='serving') \
    .bulk_infer(example_provider_component=ftfx.input_builders.from_csv(
        uri=os.path.join(current_dir, 'to_infer'),
        name='bulk_infer_example_gen'
    )) \
```

- All these are provided components by TFX
- User code is just the essentials:
  - Preprocessing
  - Data Check Thresholds
  - Tuning / Training
  - Model Evaluation Thresholds
  - Infrastructure Validation
- Statistics Generation => Automatic
- Schema Inference => Automatic
- Data Anomalies => Semi Automatic

(https://github.com/ntakouris/fluent-tfx) or view the TFX examples directly

The actual TFX API is more verbose and flexible

```python
# TODO(b/137289334): rename this as simple after DAG visualization is done.
def _create_pipeline(pipeline_name: Text, pipeline_root: Text, data_root: Text,
                     module_file: Text, serving_model_dir: Text,
                     metadata_path: Text,
                     beam_pipeline_args: List[Text]) -> pipeline.Pipeline:
  """Implements the chicago taxi pipeline with TFX."""
  examples = external_input(data_root)

  # Brings data into the pipeline or otherwise joins/converts training data.
  example_gen = CsvExampleGen(input=examples)

  # Computes statistics over data for visualization and example validation.
  statistics_gen = StatisticsGen(examples=example_gen.outputs['examples'])

  # Generates schema based on statistics files.
  schema_gen = SchemaGen(
      statistics=statistics_gen.outputs['statistics'],
      infer_feature_shape=False)

  # Performs anomaly detection based on statistics and data schema.
  example_validator = ExampleValidator(
      statistics=statistics_gen.outputs['statistics'],
      schema=schema_gen.outputs['schema'])

```

(https://github.com/tensorflow/tfx/tree/master/tfx/examples/chicago_taxi_pipeline)

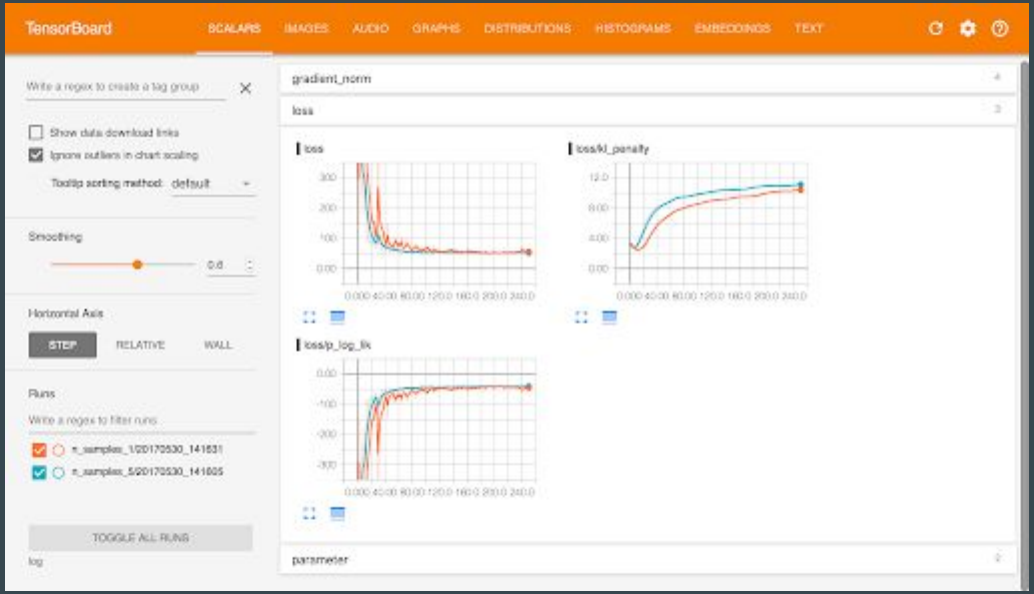
Scale? Automatic and Elegant

```
def preprocessing_fn(inputs: Dict[Text, Any]) -> Dict[Text, Any]:
    outputs = {}
    for feat in DENSE_FEATURES:
        outputs[f'{feat}_xf'] = tft.scale_to_z_score(inputs[feat])

    for feat in BINARY_FEATURES:
        outputs[feat] = inputs[feat]

    outputs[LABEL_KEY] = inputs[LABEL_KEY]
    return outputs
```

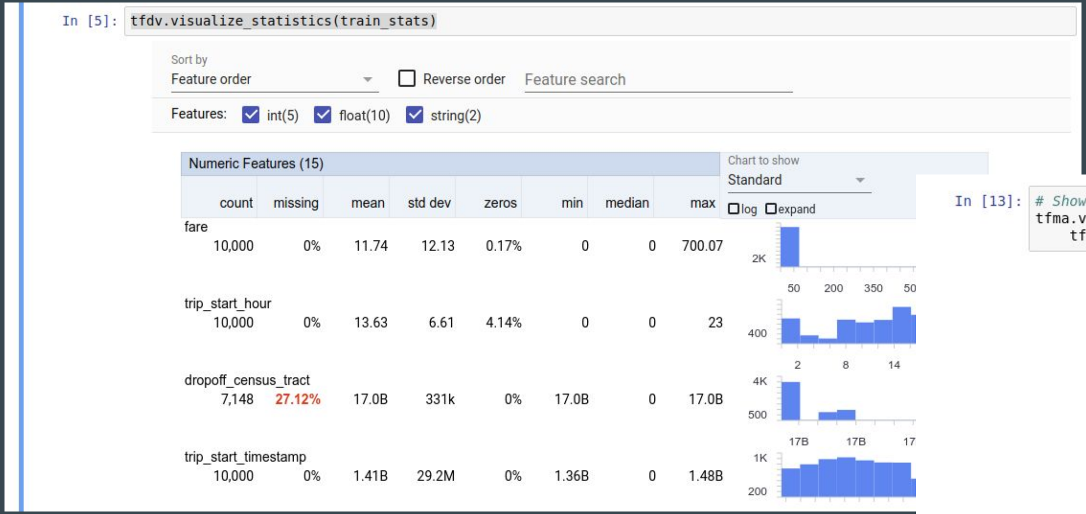
A single line of code can spin up multiple worker nodes to do computations in a parallel, map-reduce fashion!

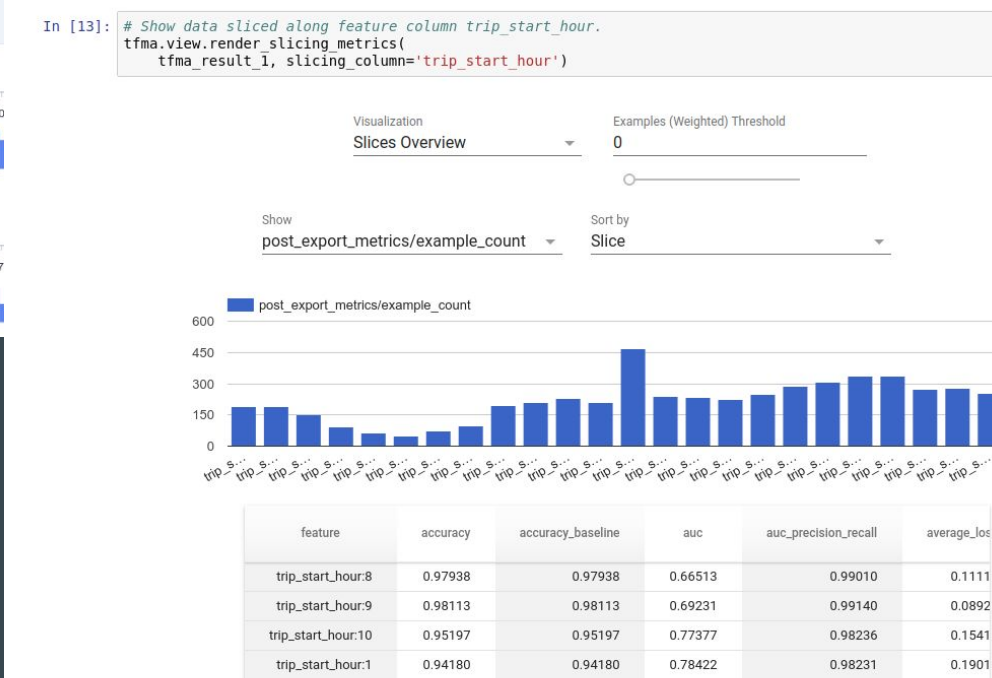
Visualizations and Experiment Tracking Tools Included

- Tensorboard for training logs
- Tensorflow Data Validation for input data inspections
- Tensorflow Model Analysis for model performance evaluation



More Visualizations





* Demo Time *

Model = Predicts if taxi trip passenger is going to tip a great amount

Data In = Trip Time, Trip Locations, Taxi Company, Etc

Data Out = big tipper probability (0 ~ 1)

Questions ?

Thank you :)

- Join WGSD at slack and facebook