Per Entity Training Pipelines in Apache Beam

Jasper Van den Bossche
ML6
About ML6

We are a group of AI and machine learning experts building custom AI solutions.

Amongst our engineers we have several Apache Beam contributors.
Agenda

● Development of ML applications
  ○ What is training?
  ○ What is MLOps?

● What does per entity training mean?
  ○ Training multiple models rather than a single model?
  ○ Why use a per entity strategy

● Example per entity training pipeline

● Bonus: Using trained models in a RunInference pipeline
What is machine learning model training?
What is machine learning model training?

```python
def contains_firefly():
    ...
```

Writing logic to detect the Beam mascot is almost impossible
What is training a machine learning model?
What is training a machine learning model?
How are machine learning applications built and deployed?
MLOps: Level 1
MLOps: Level 2

1. Model analysis
2. Source code
3. CI: Build, test, & package pipeline components
4. Experimentation/development/test staging/preproduction/production
5. Model registry
6. Performance monitoring

CD: Pipeline deployment

CD: Model serving

Data analysis
Orchestrated experiment
Source repository

Feature store

Trigger

ML metadata store

Prediction service
What is per entity training?
Example: Building multilingual chatbot

- Guten Tag!
- 안녕하세요!
- Bonjour!
What is per entity training?

Multilingual Large Language Model

Dutch Language Model

Spanish Language Model

English Language Model

Italian Language Model
Example: Detect production defects using sensor data
Example: Detect production defects using sensor data

Sensor 2

Sensor 4

Cap not mounted properly

No defect found

Sensor 1

Sensor 3

Sensor 4

Component 1 burnt

No defect found
Why use a per entity strategy?
Reduce Model Infrastructure Requirements

CPU Machine

Lightweight GPU

GPU Cluster
Faster training & inference

- Dutch Model
- German Model
- Portuguese Model
- Multilingual Large Language Model
Address fairness and bias
Easier to detect problems
Simpler models have the following advantages:

- Faster training & inference
- Easier debugging
- Less powerful hardware required
- Easier to address bias
But there is one big problem: How do I manage the training of all of these models?
Manage training pipelines

- schedule.csv
- logs.json
- 1026 hPa → Model 1
- 1016 hPa → Model 2
- 56°C 59°C 57°C 61°C → Model 1
The solution? Apache Beam!

- Apache Beam can handle *streaming* and *batch data*
- Apache Beam can easily *prepare data* for training
- Apache Beam can run on different *runners* depending on the model’s *requirements*
- *Abstraction* in ML libraries allows us to train models with few lines of code
Let’s look at an example of a per entity training pipeline
## Predicting incomes per education level

<table>
<thead>
<tr>
<th>Age</th>
<th>Workclass</th>
<th>Education</th>
<th>Marital Status</th>
<th>Occupation</th>
<th>Relationship</th>
<th>Race</th>
<th>Sex</th>
<th>Hours per Week</th>
<th>Native Country</th>
<th>Compensation</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Private</td>
<td>11th</td>
<td>Never-married</td>
<td>Machine-op-inspct</td>
<td>Own-child</td>
<td>Black</td>
<td>Male</td>
<td>40</td>
<td>USA</td>
<td>&lt;=50K.</td>
</tr>
<tr>
<td>38</td>
<td>Private</td>
<td>HS-grad</td>
<td>Married-civ-spouse</td>
<td>Farming-fishing</td>
<td>Husband</td>
<td>White</td>
<td>Male</td>
<td>50</td>
<td>USA</td>
<td>&lt;=50K.</td>
</tr>
<tr>
<td>28</td>
<td>Local-gov</td>
<td>Assoc-acdm</td>
<td>Married-civ-spouse</td>
<td>Protective-serv</td>
<td>Husband</td>
<td>White</td>
<td>Male</td>
<td>40</td>
<td>USA</td>
<td>&gt;50K.</td>
</tr>
<tr>
<td>44</td>
<td>Private</td>
<td>Some-college</td>
<td>Married-civ-spouse</td>
<td>Machine-op-inspct</td>
<td>Husband</td>
<td>Black</td>
<td>Male</td>
<td>40</td>
<td>USA</td>
<td>&gt;50K.</td>
</tr>
<tr>
<td>18</td>
<td>?</td>
<td>Some-college</td>
<td>Never-married</td>
<td>?</td>
<td>Own-child</td>
<td>White</td>
<td>Female</td>
<td>30</td>
<td>USA</td>
<td>&lt;=50K.</td>
</tr>
</tbody>
</table>
Pipeline overview

- Load Data
- Clean Data
- Group per Education Level
- Train Models
- Save Models
Split data per education level

<table>
<thead>
<tr>
<th>Age</th>
<th>Occupation</th>
<th>Education</th>
</tr>
</thead>
<tbody>
<tr>
<td>29y</td>
<td>Accountant</td>
<td>Bachelor</td>
</tr>
<tr>
<td>31y</td>
<td>Engineer</td>
<td>Master</td>
</tr>
<tr>
<td>54y</td>
<td>Plumber</td>
<td>Bachelor</td>
</tr>
<tr>
<td>37y</td>
<td>Server</td>
<td>High School</td>
</tr>
<tr>
<td>47y</td>
<td>Barista</td>
<td>High School</td>
</tr>
<tr>
<td>22y</td>
<td>Cashier</td>
<td>Bachelor</td>
</tr>
</tbody>
</table>


data for 29y, 54y, 31y, 37y, and 47y are shown with their respective education levels.
Train model per dataset
with beam.Pipeline(options=pipeline_options) as pipeline:
  _ = (  
    pipeline | "Read Data" >> beam.io.ReadFromText(known_args.input)  
             | "Split data to make List" >> beam.Map(lambda x: x.split(','))  
             | "Filter rows" >> beam.Filter(custom_filter)  
             | "Create Key" >> beam.ParDo(CreateKey())  
             | "Group by education" >> beam.GroupByKey()  
             | "Prepare Data" >> beam.ParDo(PrepareDataforTraining())  
             | "Train Model" >> beam.ParDo(TrainModel())  
             | "Save" >> fileio.WriteToFiles(path=known_args.output,
                                          sink=ModelSink()))
def custom_filter(element):
    return len(element) == 15 and '?' not in element \
    and ' Bachelors' in element or ' Masters' in element \
    or ' Doctorate' in element
Step 1: Data preparation

class PrepareDataforTraining(beam.DoFn):
    def process(self, element, *args, **kargs):
        key, values = element

        # Convert to dataframe
        df = pd.DataFrame(values)
        last_ix = len(df.columns) - 1
        X, y = df.drop(last_ix, axis=1), df[last_ix]

        # select categorical and numerical features
        cat_ix = X.select_dtypes(include=['object', 'bool']).columns
        num_ix = X.select_dtypes(include=['int64', 'float64']).columns

        # label encode the target variable to have the classes 0 and 1
        y = LabelEncoder().fit_transform(y)

        yield (X, y, cat_ix, num_ix, key)
class TrainModel(beam.DoFn):

    def process(self, element, *args, **kwargs):
        X, y, cat_ix, num_ix, key = element
        steps = [('c', OneHotEncoder(handle_unknown='ignore'), cat_ix),
                 ('n', MinMaxScaler(), num_ix)]

        # one hot encode categorical, normalize numerical
        ct = ColumnTransformer(steps)

        # wrap the model in a pipeline
        pipeline = Pipeline(steps=[('t', ct), ('m', DecisionTreeClassifier())])
        pipeline.fit(X, y)

        yield (key, pipeline)
class ModelSink(fileio.FileSink):
    def open(self, fh):
        self._fh = fh

    def write(self, record):
        _, trained_model = record
        pickled_model = pickle.dumps(trained_model)
        self._fh.write(pickled_model)

    def flush(self):
        self._fh.flush()
Extending the pipeline

... → Train Models → Calculate Metrics
class EvaluateModel(beam.DoFn):
    def __init__(self, model_uri):
        file = FileSystems.open(model_uri, 'rb')
        self.model = pickle.load(file)

    def process(self, element, *args, **kwargs):
        inputs, labels = element
        predictions = self.model.predict(inputs)
        accuracy = sklearn.metrics.accuracy_score(y_pred=predictions,
                                                 y_true=labels)
        f1 = sklearn.metrics.f1_score(y_pred=predictions, y_true=labels)
        recall = sklearn.metrics.recall_score(y_pred=predictions, y_true=labels)

        file = FileSystems.open(f'model_uri_metrics', 'wb')
        file.writelines([f'accuracy: {accuracy}', f'f1: {f1}', f'recall: {recall}'])
How does this pipeline fit in the MLOps architecture?
Let’s try out our model using the RunInference transform
Bonus: Inference in Apache Beam
Apache Beam is more and more becoming technology that can be used in advanced MLOps setups.

Per entity strategy has several advantages:
- Requires less powerful hardware
- Faster training and inference
- Easier to address bias
- Easier to debug

Apache Beam is a perfect candidate for per entity training pipelines thanks to:
- Excellent for data preprocessing and preparation
- Different runners depending on model requirements
- Abstraction in ML libraries that make it easy to train a model
QUESTIONS?

https://www.linkedin.com/in/jasper-van-den-bossche/
https://github.com/jaxpr
https://www.ml6.eu/