Ariadne - The next generation

Idea

- NLP Model server that can also train interactive models
- ML server that exposes popular Python ML libraries like sklearn, spacy, spark NLP, …

Please describe:

- Lifecycle of a model: how does it get created, how is it identified, what happens during its lifetime, when does it get deleted

Use Cases

INCEpTION external recommender

Richard his work (Java -> Python)

ML model server

- Model server that can serve and train models, e.g. sklearn via http

Model Centric

- someone annotates documents in inception using an external recommender
- a separate process also accesses the external recommender to make predictions
- if they think that these other predictions are not good enough, import matching files into inception, annotate them there to improve suggestions

Other annotation tools

- Add this to other tools

Requirements

Functional requirements

- Concurrent training
- Concurrent prediction
- Prediction automatically switches to new model once training is complete
- “Batching” - i.e. prediction switches only at the end of the batch - it is ensured that all documents in the batch are predicted with the same model
- Lightweight self-contained solution
- Easy to interface with any Python ML library / model
- Easy to parse/generate data model in the communication (e.g. JSON)

- Easy installation “pip install ariadne”
- Ability to support different content types for the data (e.g. ability to send plain text, return a simple JSON, send XMI CAS, get back Apache Arrow)
- Discoverable endpoints (info about version, attached models and their types)
- Ability to store arbitrary metadata per model (e.g. for which user, client, project, whatever the model was generated)
- Ability to pass training parameters to the ML library (the metadata feature can be “abused” for that)
- Ability to “update” a model - i.e. send new data and incorporate it into an existing model without full retraining (online learning) - iff supported by the integrated ML library
- Models can be deleted on request or after X time has elapsed
- Model training status can be queried

Scalability requirements
- Server retains cache of training data and client can send updates/deletes to the cache
- Can predict on part of a document

Technical requirements
- Communication via HTTP/Websocket ? Or grpc/apache arrow or so
- Model not repeatedly loaded per worker or model is only loaded on demand
- Async training/prediction, either via async http library or callbacks/webhooks

Other tools

Tensorflow server

https://www.tensorflow.org/tfx/serving/api_rest

Ludwig

https://github.com/ludwig-ai/ludwig

Apache Spark NLP

TorchServe

https://pytorch.org/serve/

https://www.aclweb.org/anthology/2020.iwltp-1.10.pdf

Lapps Grid

Ray serve

https://medium.com/distributed-computing-with-ray/how-to-scale-up-your-fastapi-application-using-ray-serve-c9a7b69e786

API

GET	/ping	Use this to check server connection
PUT	/dataset/<dataset_id>	Create dataset
GET	/dataset/<dataset_id>	Returns list of document names/ids
PUT	/dataset/<dataset_id>/<document_id>	Add document to dataset
DELETE	/dataset/<dataset_id>	Delete dataset
GET	/model	Get info about all models
GET	/mode/<model_id>	Get info about specific model
POST	/model/<model_id>/train/<dataset_id>	Train model on dataset
POST	/model/<model_id>/predict/	Predict for document in request
POST	/model/<model_id>/predict/<document_id>	Predict with model on specified doc
DELETE	/model/<model_id>/<user_id>	Delete trained user model

Document format for training and predicting

```
{
        “metadata”: {
                “token_type”: "cassis.token",
                 “target_type”: “ner”,
                “target_feature”: “label”,
                ...
        },
        "text": "Joe waited for the train . The train was late .",
        “data”: {
           "cassis.token": [
             {
                 “begin”: 0,
                 “end”: 3,
             },
             ...
           ],
           “cassis.ner”: [
             {
                 “begin”: 0,
                 “end”: 3,
                 “label”: “PER”
             },
             ...
           ]
        }
}
```

Require accept header so that we can in the future use other formats like arrow or bson

No need for complicated type systems, feature definitions, …
I dont want to use cassis/uima if I do not need to, we can add converter for XMI and other popular format to ariadne json

Flows

Pretrained model without training

Only predict with sending documents in body and maybe getting info about models

Model without training

1. Create dataset
2. Predict on document in dataset
3. When adding new data on client side, check which documents are already on the server and only add new ones