Machine Learning Systems Design

Deployment and Monitoring

Lecture 17: Model Serving

CE 40959 Spring 2023
Ali Zarezade
SharifMLSD.github.io

Agenda

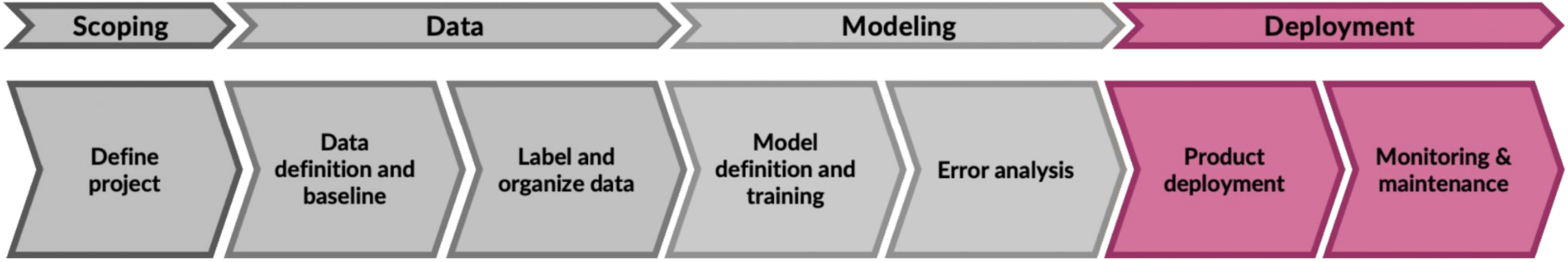
1. Model Deployment Patterns
2. Model Deployment Strategies
3. Automatic Deployment
4. Model Deployment Best Practices

1. Model Deployment Patterns

Model deployment

Deploying a model means to make it available for accepting queries generated by the users of the production system.

Once the production system accepts the query, the latter is transformed into a feature vector. The feature vector is then sent to the model as input for scoring. The result of the scoring then is returned to the user.



Model deployment patterns

A model can be deployed following several patterns:

- statically, as a part of an installable software package
- dynamically on the user’s device,
- dynamically on a server
- via model streaming

Static deployment

Prepare an installable binary of the entire software. The model is packaged as a resource available at the runtime.

Static deployment

Pros:

- Software has direct access to the model (fast execution time)
- User data doesn't have to be uploaded to the server at the time of prediction (saves time and preserves privacy)
- Model can be called when the user is offline
- Software vendor doesn't have to care about keeping the model operational; it becomes the user's responsibility.

Static deployment

Cons:

- Hard to separate machine learning code from application code
- Hard to upgrade model without upgrading application
- Computational requirements may limit deployment options (GPU access)

Dynamic deployment on user device

Similar to static, but the model is not part of the binary code of the application.

Dynamic deployment on user device

It can be achieved by:

- deploying model parameters
    - the model file only contains the learned parameters, while the user's device has installed a runtime environment for the model (TensorFlow Lite: TF models, Apple Core ML: sklearn, keras, xgboost)

Dynamic deployment on user device

It can be achieved by:

- deploying model parameters
- deploying a serialized object
  - It uses a model file as a serialized object that the app can deserialize. It avoids runtime dependencies but makes updates large and costly.

Dynamic deployment on user device

It can be achieved by:

- deploying model parameters
- deploying a serialized object
- deploying to the browser
    - TensorFlow.js, have versions that allow to train and run a model in a browser, by using JavaScript as a runtime.

Dynamic deployment on user device

Pros:

- calls to the model will be fast for the user
- reduce load on servers
- better separation of concerns
- easier model updates
- adaptive model selection based on compute resources

Dynamic deployment on user device

Cons:

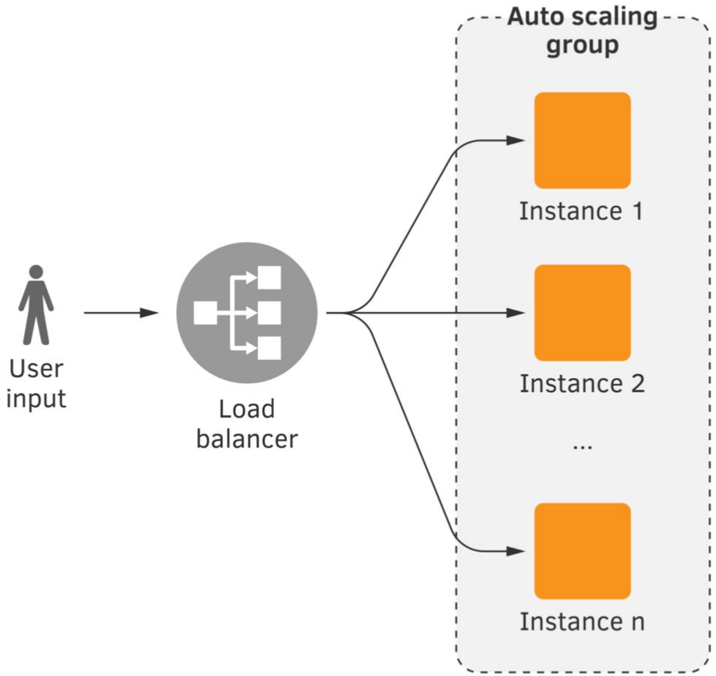
- More bandwidth and startup time for the browser-based deployment.
- Harder to update the model and keep it consistent across users.
- Easier for third parties to analyze and manipulate the model.

Dynamic deployment on a server (VM)

The most frequent deployment pattern is to place the model on servers, and make it available as a REST API in the form of a web service, or gRPC service.

Dynamic deployment on a server (VM)

- A web service receives user requests with input data and calls the machine learning system to get predictions.
- The predictions are returned as JSON or XML strings.
- To handle high load, multiple web service instances run on virtual machines in parallel.
- A load balancer distributes the requests among the instances.



Dynamic deployment on a server (VM)

Pros:

- Simple and familiar software system.

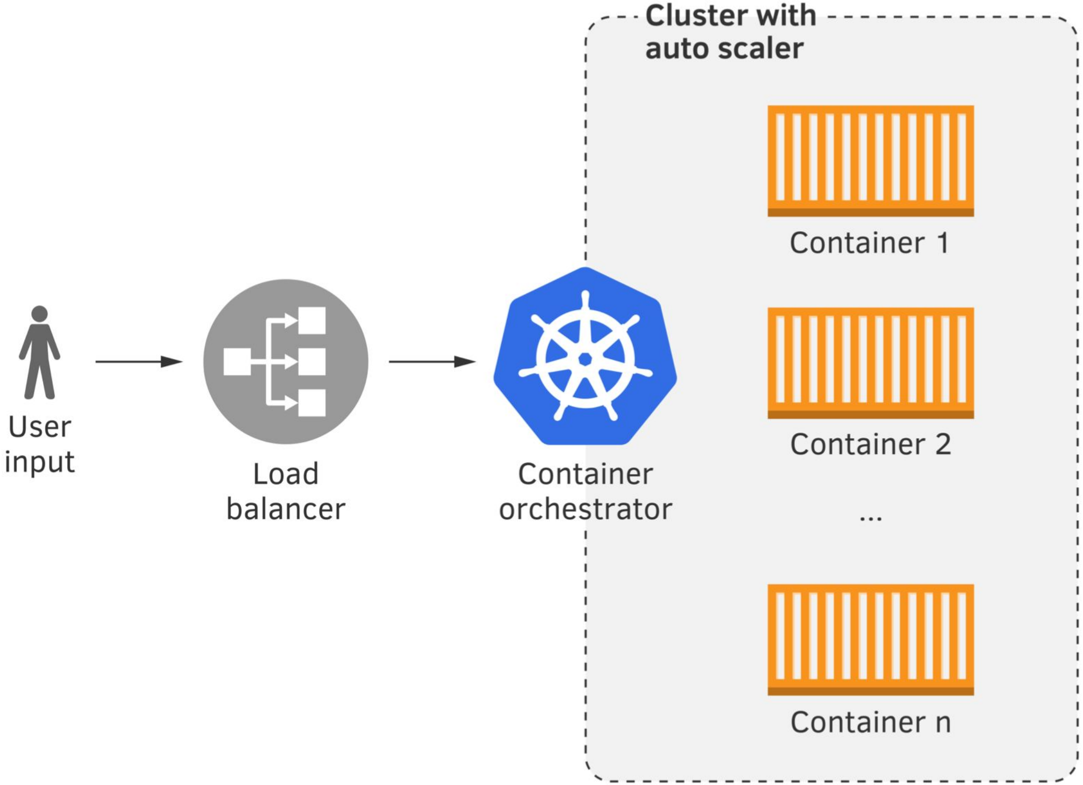
Cons:

- Need to maintain servers (physical or virtual).
- Network latency and computational overhead.
- Relatively higher cost.

Dynamic deployment on a server (container)

- A container is like a virtual machine, but it shares the operating system with other containers on the same machine.
- The machine learning system and the web service are installed inside a container (e.g., Docker).
- A container-orchestration system (e.g., Kubernetes) runs the containers on a cluster of servers.
- The cluster can be scaled up or down automatically or manually.

Dynamic deployment on a server (container)



Dynamic deployment on a server (container)

Pros:

- More resource-efficient and flexible than virtual machines.

Cons:

- Still need to maintain servers (physical or virtual).
- Still have network latency and computational overhead.
- More complicated and requires expertise.

Dynamic deployment on a server (serverless)

- A way of running machine learning systems on cloud platforms without managing servers or resources
- Requires a zip archive with code, model, and entry point function
- Provides an API to submit inputs and receive outputs

AWS Lambda

Dynamic deployment on a server (serverless)

Pros:

- Supports multiple programming languages and dependencies
- Highly scalable and supports synchronous and asynchronous modes
- Cost-efficient: only pay for compute-time
- Simplifies canary deployment: test new code on a small group of users
- Easy rollbacks: switch back to previous version by replacing zip archive

Dynamic deployment on a server (serverless)

Cons:

- Has limits on execution time, zip file size, and RAM
- No GPU access for deep models

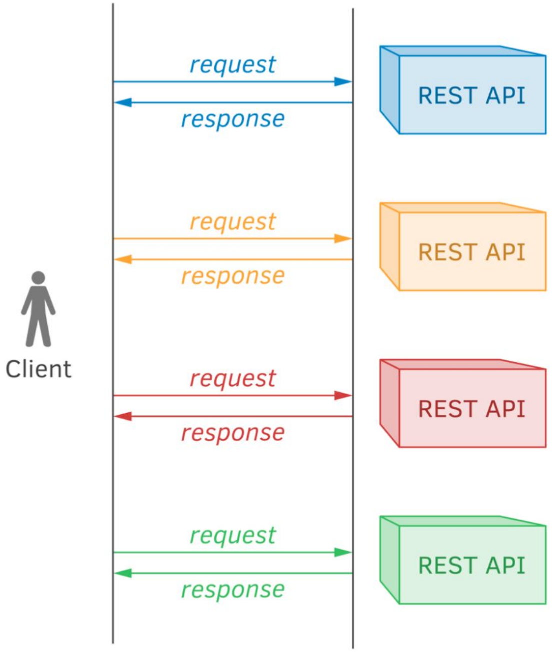
Dynamic deployment on a server (model streaming)

- Model streaming is a deployment pattern that can be seen as an inverse to the REST API
- In REST API, the client sends a request to the server and waits for a response (a prediction)
- In streaming, the client sends a request to a stream-processing application and receives update events as they happen
- The stream-processing application has a data processing topology that defines the data flow and transformations

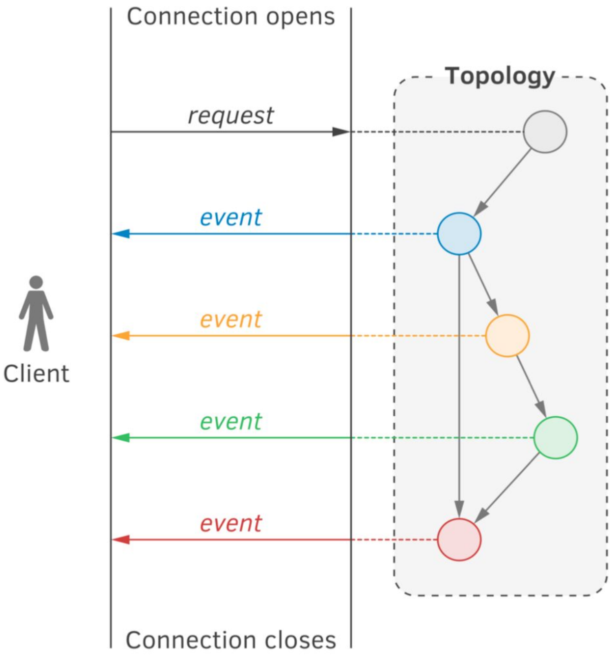
Dynamic deployment on a server (model streaming)

- **Stream-processing engines (SPEs)** are frameworks that run on their own clusters and distribute the data processing load among the available resources (Apache Storm, Apache Spark, and Apache Flink)
- **Stream-processing libraries (SPLs)** are libraries that can be integrated with available resources, such as virtual or physical machines, or a container orchestrator (Apache Samza, Apache Kafka Streams, and Akka Streams)

Dynamic deployment on a server (model streaming)



(a) REST API



(b) streaming

Dynamic deployment on a server (model streaming)

REST API vs Streaming:

- REST APIs are usually employed to let clients send ad-hoc requests that don't follow a certain frequently-repeated pattern
- It's the best choice when the client wants the liberty of deciding what to do with the API response
- Streaming-based applications provide better resource-efficiency, lower latency, security, and fault-tolerance when each request of the client is typical, undergoes a certain pattern of transformations, and always results in the same actions

2. Model Deployment Strategies

Single deployment

It is the simplest one. Once you have a new model, you serialize it into a file, and then replace the old file with the new one. You also replace the feature extractor, if needed.

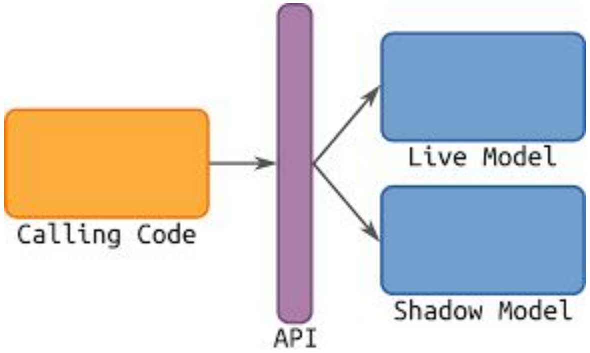
How to deploy on users device, cloud or physical server?

Single deployment

It is also the riskiest strategy. If the new model or the feature extractor contains a bug, all users will be affected.

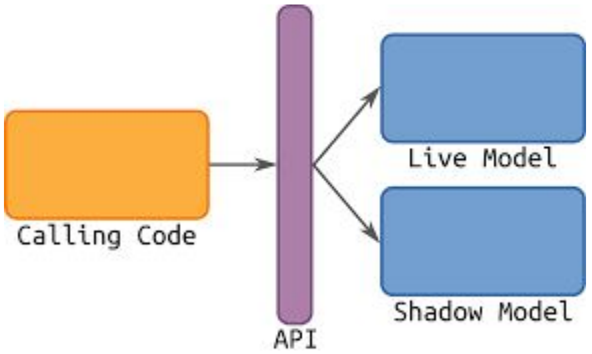
Silent deployment

- Deploy new and old versions of model and feature extractors in parallel
- Only log predictions from new version, don't show them to user
- Analyze predictions later to check for bugs

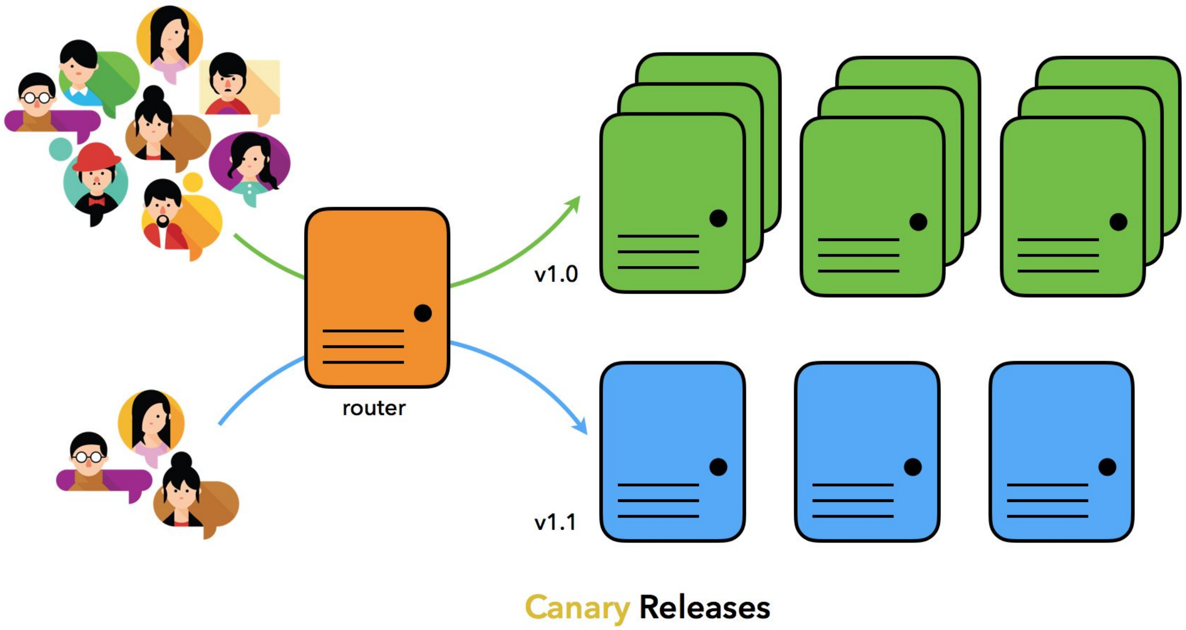


Silent deployment

- Pros: No user impact, more time to test new model
- Cons: More resource consumption, hard to evaluate without user feedback



Canary deployment



Canary deployment

- Pushes the new model version and code to a small fraction of users, while keeping the old version running for most users.
- Contrary to the silent deployment, canary deployment allows validating the new model's performance, and its predictions' effects.
- Contrary to the single deployment, canary deployment doesn't affect lots of users in case of possible bugs.

Canary deployment

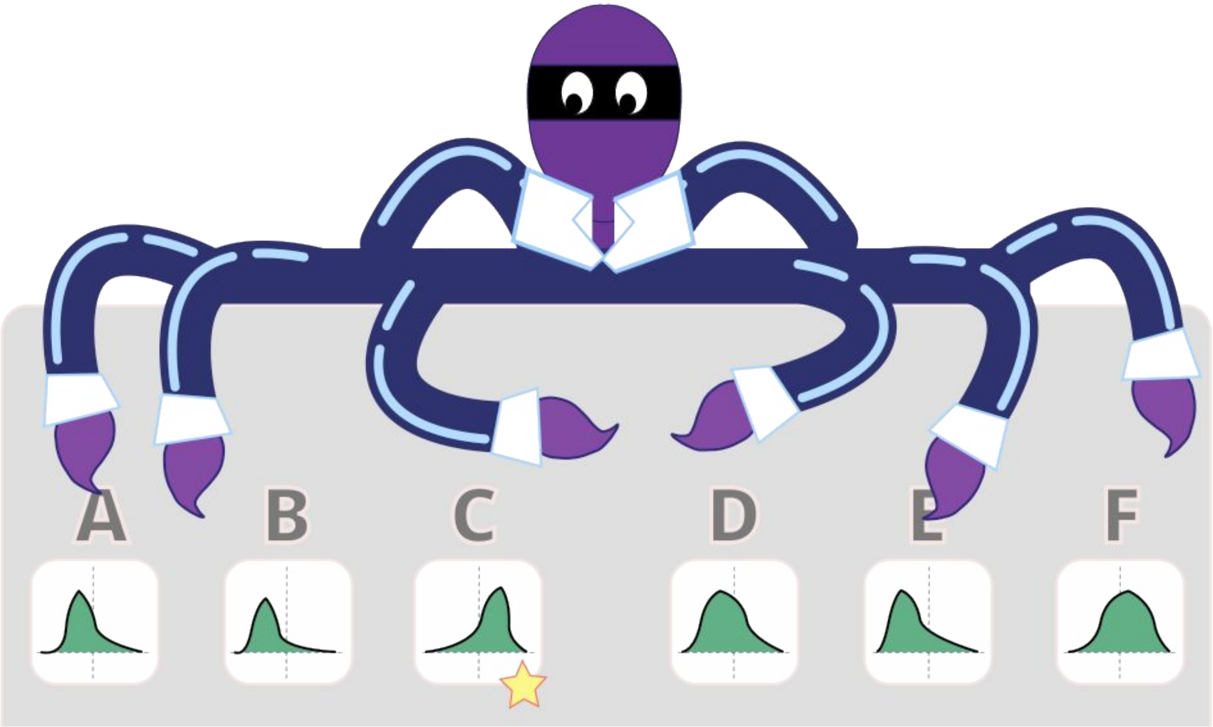
Pros:

- Doesn’t affect lots of users in case of possible bugs.

Cons:

- Accept more complexity of having multiple model versions deployed
- Hard to spot rare errors: If you deploy the new version to 5% of users, and a bug affects 2% of users, then you have only 0.1% chance that the bug will be discovered.

Multi-armed bandits deployment



Multi-armed bandits deployment

- After the convergence of the MAB algorithm, most of the time, all users are routed to the software version running the best model.
- The MAB algorithm, thus, solves two problems, online model evaluation and model deployment simultaneously.

3. Automated Deployment

Automated deployment

The model is an important asset, but it’s never delivered alone. There are additional assets for production model testing that ensure the model is not broken.

End-to-end test

Only deploy a model in production when it’s accompanied with the following assets:

- an **end-to-end set** that defines model inputs and outputs that always works
- a **confidence test set** that correctly defines model inputs and outputs, and is used to compute the value of the metric
- a **performance metric** whose value will be calculated on the confidence test set by applying the model to it
- the **range of acceptable values** of the performance metric.

If evaluations failed, the model should not be served to the client.

Version Sync

- Keep training data, feature extractor, and model versions in sync
- Update versions whenever any of them changes
- Automate deployment of new model version with a script
- Fetch model and feature extractor from repositories and copy to production
- Apply model to end-to-end and confidence test data
- Roll back deployment if prediction error or metric value is unacceptable

Model version metadata

Each model version must be accompanied with the following code and metadata:

- the name and the version of the library or package used to train the model
- if Python was used to build the model, then requirements.txt (or, alternatively, a Docker image name pointing to a specific path on Docker Hub or in your Docker registry)
- the name of the learning algorithm, and names and values of the hyperparameters
- the list of features required by the model
- the list of outputs, their types, and how the outputs should be consumed
- the version and location of the data used to train the model
- the version and location of the validation data used to tune model hyperparameters
- the model scoring code that runs the model on new data and outputs the prediction.

The metadata and the scoring code may be saved to a database or to a JSON/XML text file.

Model version metadata

For audit purposes, the following information must also accompany each deployment:

- who built the model and when
- who and when made the decision of deploying that model, and based on what grounds
- who reviewed the model for privacy and security compliance purposes.

4. Model Deployment Best Practices

Algorithmic efficiency

- Optimize your algorithms for best time and space complexity.
- Avoid using loops whenever possible, and use NumPy or similar tools.
- Use appropriate data structures, list, set, dict (hash table).
- If you need to iterate over a vast collection of elements, use Python generators that create a function returning one element at a time, rather than all elements at once.
- Use the cProfile package in Python to find code inefficiencies.
- Boost the speed by using multiprocessing package in Python to run computations in parallel; or use a distributed processing framework such as Apache Spark.
- Use PyPy, Numba or similar tools to compile your Python code into fast, optimized machine code.
- Serve the parts of your code that need GPU on GPU servers and the rest on regular CPU servers.

Caching

- Cach resource-consuming functions which frequently called with the same parameter values
- The simplest cache may be implemented in the application itself, like lru_cache decorator in python which wrap a function with a memoizing callable that saves up to the maxsize most recent calls.
- In large scale production systems, engineers employ general purpose scalable and configurable cache solutions such as Redis or Memcached.

Delivery format for model and code

Serialization is the most straightforward way to deliver the model and the feature extractor code to the production environment (Python pickle, Scikit-learn joblib)

If the production code is written in a compiled language (Java or C/C++), and ML Engineers built models using Python, there are three options to deploy for production:

- rewrite the code in a compiled, production-environment programming language
- use a model representation standard such as PMML or PFA, or
- use a specialized execution engine such as MLeap

PMML

The Predictive Model Markup Language (PMML) is an XML-based predictive model interchange format that provides a way for data analysts to save and share models between PMML-compliant applications.

```
<DataDictionary numberOfFields=“3”> <DataField name=“Sepal_Length” optype=“continuous”
dataType=“double”/> <DataField name=“Petal_Length” optype=“continuous”
dataType=“double”/> <DataField name=“Species” optype=“categorical” dataType=“string”>
<Value value=“setosa”/> <Value value=“versicolor”/> <Value value=“virginica”/>
</DataField> </DataDictionary> <RegressionModel modelName=“Linear_Regression_Model”
functionName=“regression” algorithmName=“linearRegression”> <MiningSchema> <MiningField
name=“Sepal_Length” usageType=“active”/> <MiningField name=“Petal_Length”
usageType=“active”/> <MiningField name=“Species” usageType=“target”/> </MiningSchema>
<RegressionTable intercept=“-0.24872358602445785”> <NumericPredictor name=“Sepal_Length”
coefficient=“-0.20594816896319375”/> <NumericPredictor name=“Petal_Length”
coefficient=“0.22282886310305097”/> </RegressionTable> </RegressionModel>
```

PFA

- Portable Format for Analytics, is a standard for representing both statistical models and data transformation engines.
- PFA allows us to easily share models and machine learning pipelines across heterogeneous systems and provides algorithmic flexibility.
- Models, pre/post-processing transformations are all functions that can be arbitrarily composed, chained, or built into complex workflows.
- PFA has a form of a JSON or a YAML configuration file.

```
{"input":"double","output":"double","action":[{"+":[{"*":["input",3.14]},2.718]}]}
```

Mleap

A tool that can run and share machine learning models and pipelines in different systems. It can export and import models from Spark, scikit-learn, TensorFlow, and others using a JSON or YAML file format.

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.regression.LinearRegression
import ml.combust.mleap.spark.SparkSupport._
import resource._

// create a linear regression model
val lr = new LinearRegression().setLabelCol("label").setFeaturesCol("features")

// create a pipeline with the model
val pipeline = new Pipeline().setStages(Array(lr))

// fit the pipeline on some data
val model = pipeline.fit(trainingData)

// export the model to an MLeap Bundle
for(modelFile <- managed(new File("model.zip"))) {
  model.writeBundle.save(modelFile)
}
```

Start with a simple model

- Production deployment can be complex and require solid infrastructure
- Simple models are easier to debug and have fewer dependencies and hyperparameters
- Complex models and pipelines are more error-prone and harder to tune

Test on outsiders

- Test your model on outsiders, not just on test data
- Outsiders can be other team members, company employees, crowdsourcing, or real customers
- Testing on outsiders helps avoid personal bias and exposure to different users

Machine Learning Systems Design

Deployment and Monitoring

Next Lecture: Model Serving (cont.)

CE 40959 Spring 2023
Ali Zarezade
SharifMLSD.github.io