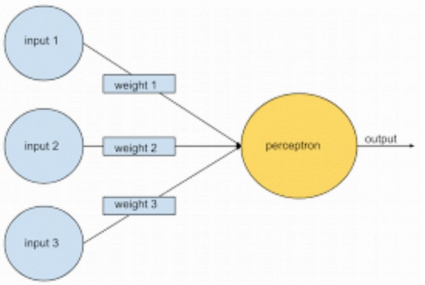
ML
r&d to production

Andre da Palma, Kare - Engineering

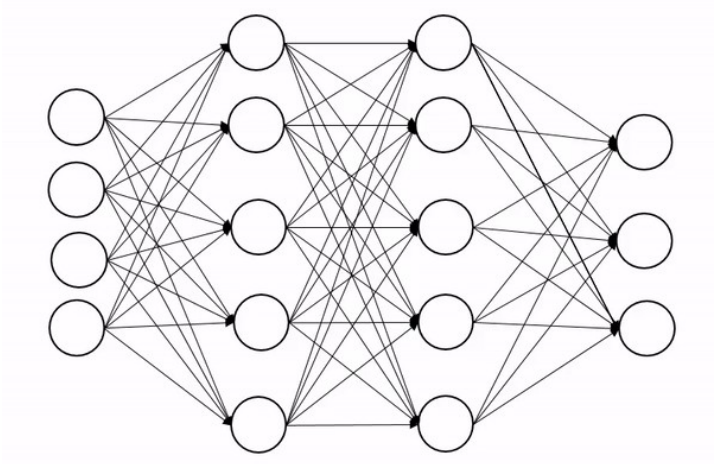
What is a ML model?

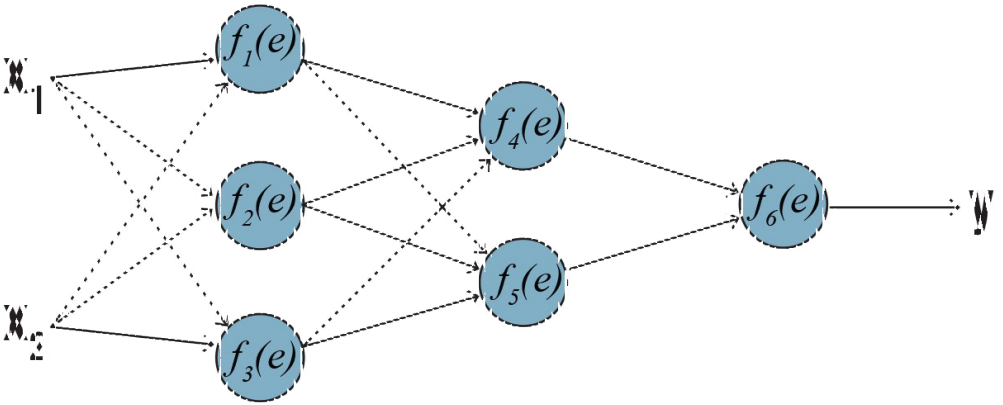
“Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task”



Objective

- Input (X) / Output (Y)
- Adjust Weights(W) in order to reduce error
- Adjust model hyperparameters
- Export best set of Weights(LTM)





Process overview

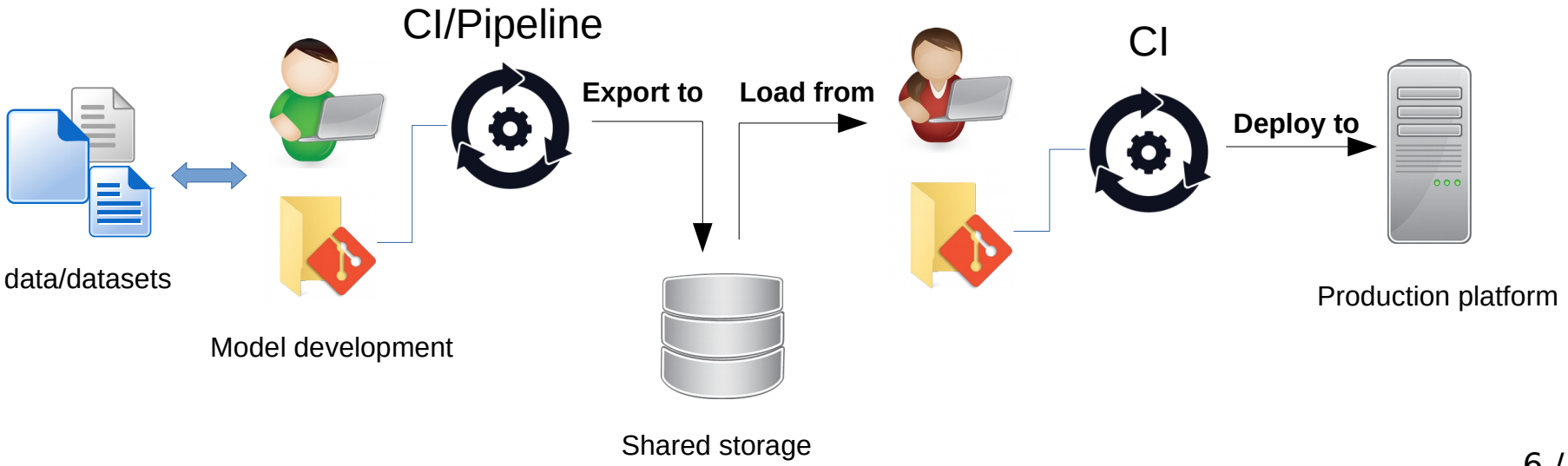
- Find and curate a dataset
- Create a model (tensorflow, torch, keras etc...)
- Train model
- Evaluate and record results
- Export model (preferably in universal format)
- Load and serve model export

Common issues

- Lack of a systematic model development process
- Code fragmentation during model development
- Model and dataset versioning
- Not clear how to use and monitor the model
- Hard to create a build pipeline
- Code quality neglected during early stages

Teams and process overview

- R&D (produce and train models)
- Platform/Back-end (run/integrate the with a application)
- DevOps (establish CD/CI processes)



R&D projects framework

- Structure project as a framework
- Create top level classes for main concepts (e.g: Model, Task, Datasets, Model and Task configurations)
- Models export should contain weights, architecture and hyper-parameter configurations
- Models should be re-buildable from previous exports
- Account for usage of multiple frameworks in same project
- Make it easy to hook revelant tasks on CI pipeline

R&D project structure

- Package individual projects
- Common and utils modules
- Utility commands for common tasks(e.g create new project)
- Projects with train and evaluation hooks

E.g at:

- https://github.com/NTMC-Community/MatchZoo

```
# Example
./repo/
    ./bin/
        create_project.py
    ./common/
        ./datasets/
            ./raw/
                yahoo.py
                locations.py
    ./models/
        ./arci/
            train.sh
            evaluate.sh
            model.py
        ./linear_regression/
            train.sh
            evaluate.sh
            model.py
```

R&D datasets as classes

- Datasets are under version control (git, svn, hg)
- Datasets shared between multiple models
- Changes are tracked

```
# Example
from common.datasets.raw import yahoo
```

```
# keras
from keras.datasets import cifar10
```

```
# tensorflow
import tensorflow_datasets as tfds
mnist_train = tfds.load(name="mnist", split="train")
```

R&D model class

- Consistent API for model development
- Common methods well defined
- Account for multiple ML frameworks

```
# Example
class RnDModel(object):
    def __init__(self, config):
        self.config = config

    def train():

    def evaluation():

    def predict():

    def build():

    def export():

    def load():
```

```
# Keras
from keras.models import Model
from keras.layers import Input, Dense

class MyModel(RnDModel):
    def build(self):
        a = Input(shape=(32,))
        b = Dense(32)(a)
        self.model = Model(inputs=a, outputs=b)
        return self.model
```

```
# tensorflow
class MyModel(RndModel):
    def build(self):
        self.X = tf.placeholder(tf.float32)

        self.M = tf.Variable(np.random.rand())
        self.B = tf.Variable(np.random.rand())

        # model:  Y = M*X + B
        self.model = tf.add(tf.multiply(self.M, self.X), self.B)
```

R&D model and task config

- Configuration of model structure
- Configuration of tasks(train, evaluation)
- Exportable and loadable from and to a common file format (e.g: json)

```
# example
class Config(object):
    def __init__():
        self.export_path =

    def to_json():

    def from_json():

    def export():

    def load():
```

```
# json config example
{
        ”train”: {
                ”epochs”: 10,
                ”optimizer”: “adam”
        },
        ”model”: {
                ”layers”: 10,
                ”layer_size”: 200
        }
}
```

R&D task classes

- Task entry points are important for automation
- ML pipelines depend on call consistency
- Tasks are very similar between models (almost allways)

```
# Example
class Job(object):
    def __init__(self, config):
        self.config = config
        self.job_dir =
        self.model_version =

    def run():
        # config
```

```
class Train(Job):
    def run():
        model = create_model(self.config)
        model.train()
        model.export()


class Evaluation(job):
    def run():
        model = load_model(self.config)
        results = model.evaluation()
        self.create_report(results)
```

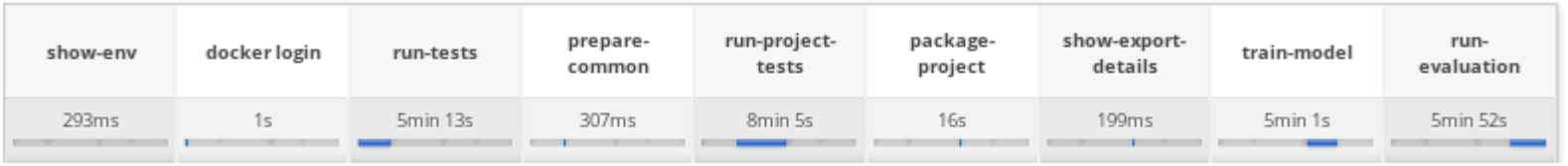
R&D report log classes

- Record key result metrics for each model task
- Save records in a database
- Keep track of model performance
- Record job configuration originated each result
- Keep historic of train and evaluations

eval_data_iterator	evaluation_configuration	model_id	model_name	model_version	prob_1	prob_5
squadtestiterator	{"length_limits": null, "training_file_num": 1, "glo...	dssm__PR_203_d08c6b3_3__0	dssm	dssm__PR_203_d08c6b3_3	0.5636160714285714	0.7469308035714286
testdataset	{"embed_size": 100, "keras_model": "gs://rnd-s...	dssm__PR_203_d08c6b3_3__0	dssm	dssm__PR_203_d08c6b3_3	0.2425579202586207	0.4284752155172414
testdatasetwithcategories	{"glove_embedding_dir": "gs://rnd-shared/pretr...	dssm__PR_218_05db674_3__0	dssm	dssm__PR_218_05db674_3	0.30845905172413796	0.5372440732758621
testdatasetwithcategories	{"alphabet": "", "slack_notify": true, "use_only_q...	dssm__PR_218_05db674_4__0	dssm	dssm__PR_218_05db674_4	0.012762661637931034	0.05744881465517242
testdatasetwithcategories	{"job_dir": "gs://rnd-shared/models/dssm/vers...	dssm__PR_218_05db674_5__0	dssm	dssm__PR_218_05db674_5	0.5356613685344828	0.7429283405172413
testdatasetwithcategories	{"reg_lambda": 0.001, "model_version": "dssm_...	dssm__PR_218_05db674_6__0	dssm	dssm__PR_218_05db674_6	0.029465247844827586	0.10833108836206896

CI build pipeline

- Similar to other build pipelines
- Pull Requests are also buildable (usefull for experimentation)
- Common stages (unit/integration test, pre-process, train and evaluate)
- Allow choice for which model pipeline to start



Model version

- Unique
- Commit IDs should be part of the version
- Consider hyper-param optimization when creating a version ID template
- Should be useful to identify and recover model and its settings

```
# example
Version template:
    {model_name}__{branch_name}__{commit}__{build_NR}__{trial_ID}
```

Model export

- Contain weights/LTM files (obviously…)
- Should contain relevant configuration (useful to know how to use the model)
- Should contain set of predictions (useful for testing)

```
[andrefsp@debian ~]$ gsutil ls gs://rnd-shared/models/dssm/versions/dssm__PR_185_97de673_89/job/export/
gs://rnd-shared/models/dssm/versions/dssm__PR_185_97de673_89/job/export/
gs://rnd-shared/models/dssm/versions/dssm__PR_185_97de673_89/job/export/config.json
gs://rnd-shared/models/dssm/versions/dssm__PR_185_97de673_89/job/export/io_conf.json
gs://rnd-shared/models/dssm/versions/dssm__PR_185_97de673_89/job/export/saved_model.pb
gs://rnd-shared/models/dssm/versions/dssm__PR_185_97de673_89/job/export/test_embeddings.csv
gs://rnd-shared/models/dssm/versions/dssm__PR_185_97de673_89/job/export/vectorizer.json
gs://rnd-shared/models/dssm/versions/dssm__PR_185_97de673_89/job/export/word_embed_vectorizer.json
gs://rnd-shared/models/dssm/versions/dssm__PR_185_97de673_89/job/export/variables/
```

Model serving

- Model embedding (load model on code)
- Model micro-services with packaging tools (e.g: docker)
- Model servers (e.g TensorServe)

Model deployment

- Loads the model form the export location
- Performs adequacy test
- Resumes normal operation
- Where model is loaded depends on model specifications (loaded on server, loaded when compiling etc...)

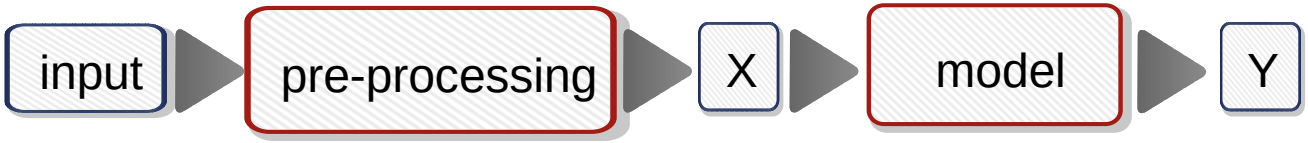
```
# golang tensorflow
import (
        tf "github.com/tensorflow/tensorflow/tensorflow/go"
)

model, err := tf.LoadSavedModel(
        ”path/to/export/”, tags, nil,
)
```

```
# golang pytorch
import (
    "github.com/orktes/go-torch"
)

module, err := torch.LoadJITModule(
        "path/to/export/model.pt",
)
```

Deployment adequacy tests

- Checks if model is being used correctly after load
- Use pre-processing examples provided on export

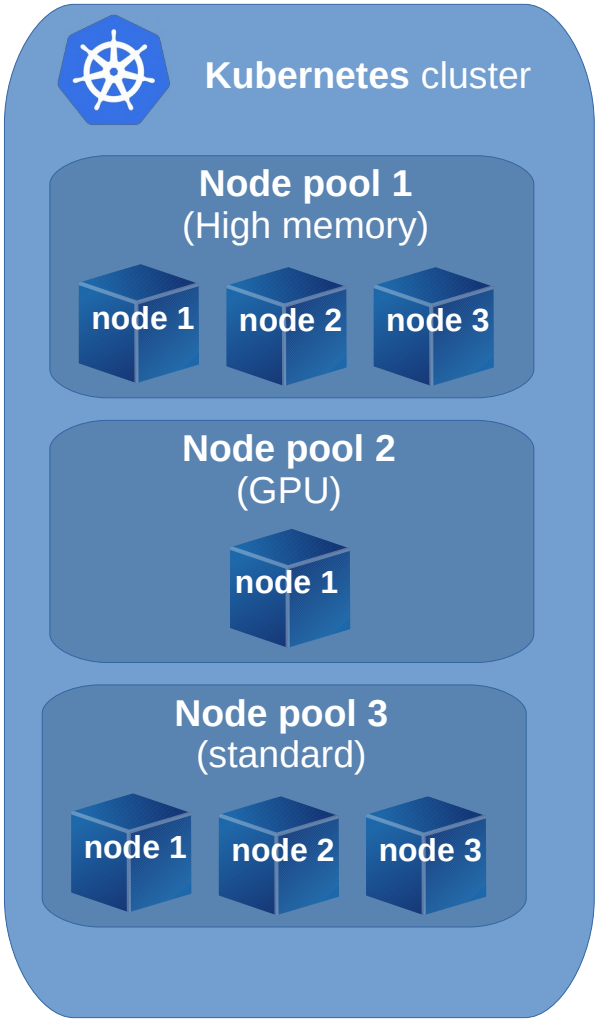
```
# adequacy_test.csv (with pre processing)

Input,Y
“normal name”,”0.87”
“strange ~ name”,”0.12”
“sentence~ requ1res preprocessing”,”0.1”
```




```
# adequacy_test.csv (Y = 2*X+1)

Input,Y
“1.0”,”3.0”
“2.0”,”5.0”
“3.0”,”7.0”
```

Deployment infra-structure

- Depends on model requirements(e.g: memory, GPU etc.)
- Container orchestration tools make it easier and flexible
- Must consider different serving strategies (embedding vs serving)



Thanks

- Model and dataset versioning tools:
  - DVC (https://dvc.org/)
  - MLFlow (https://mlflow.org/)
- Pipeline and ML flow and model serving tools:
  - Kubeflow (https://www.kubeflow.org)
  - RedisAI (https://oss.redislabs.com/redisai/)
- Check out:
  - https://www.tensorflow.org/tfx/guide/serving
  - https://pytorch.org/blog/model-serving-in-pyorch/
- Repos:
  - https://github.com/NTMC-Community/MatchZoo