ML as microservice

Illustration of running PySpark application

Two application approaches

- Build model in Spark
  1. Run model with application in Spark
  2. Export model (PMML) and run model & application in another language (Python)

In Spark: Build and save a model

- Python notebook to create your model using standard method
- Save the model, which can be loaded into a smaller Docker container running PySpark

```
def fit_model(pipelineModel, data):
    global loaded_model
    preppedDataDF =
pipelineModel.transform(data)
    lrModel =
LogisticRegression().fit(preppedDataD
F)

lrModel.write().overwrite().save("lrM
odel")
```

Docker image that runs web server in PySpark

- Create PySpark Python environment on a Docker image
  - Setup PySpark on linux
- Config Docker to start app.py, running web server
  - Runs Flask, for example
- When receives request, then runs model with given data

```
app = Flask(__name__)

@app.route('/')
def hello_world():
    # return 'Hello World!'
    return test()
```

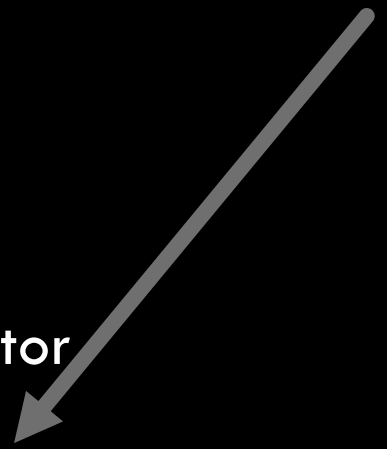
```
def run_model(pipelineModel, dataset, cols):
    preppedDataDF = pipelineModel.transform(dataset)
    lrModel = LogisticRegressionModel.load("lrModel")
    selectedcols = ["label", "features"] + cols
    predDF = preppedDataDF.select(selectedcols)
    predictions = lrModel.transform(predDF)
    selected = predictions.select("label", "prediction",
"probability", "age", "occupation")
    return selected
```

Illustration in AWS

Create & deploy a model

- Create s3 file folder for data
- Create an XGBoost container
  - Which contains the XGBoost code, in a format known by the AWS API
- Train the model
- Deploy the model, in a new container
- Send the deployed model data
- → All model and containers managed by AWS API

Create a sized instance of the XGBoost container



```
# Create the instance (Docker) with the XGBost estimator
sess = sagemaker.Session()
xgb = sagemaker.estimator.Estimator(containers[my_region],role, train_instance_count=1,
train_instance_type='ml.m4.xlarge',output_path='s3://{}/{}/output'.format(bucket_name,
prefix),sagemaker_session=sess)
xgb.set_hyperparameters(max_depth=5,eta=0.2,gamma=4,min_child_weight=6,subsample=
0.8,silent=0,objective='binary:logistic',num_round=100)
```

Train the model in the container

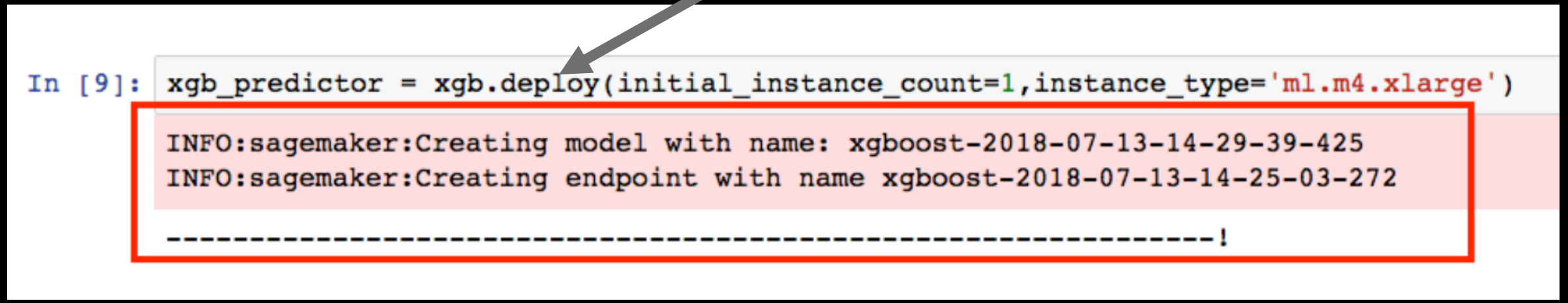
```
In [7]: xgb.fit({'train': s3_input_train})
```

```
[17:36:25] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 10 extra nodes, 14 pruned nodes, max_depth=5
[93]#011train-error:0.095314
[17:36:25] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 24 extra nodes, 30 pruned nodes, max_depth=5
[94]#011train-error:0.095314
[17:36:25] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 6 extra nodes, 24 pruned nodes, max_depth=3
[95]#011train-error:0.095314
[17:36:25] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 12 extra nodes, 30 pruned nodes, max_depth=5
[96]#011train-error:0.095279
[17:36:25] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 18 extra nodes, 12 pruned nodes, max_depth=5
[97]#011train-error:0.094828
[17:36:26] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 4 extra nodes, 22 pruned nodes, max_depth=2
[98]#011train-error:0.094863
[17:36:26] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 12 pruned nodes, max_depth=5
[99]#011train-error:0.094759


2019-08-15 17:36:34 Uploading - Uploading generated training model
2019-08-15 17:36:34 Completed - Training job completed
Billable seconds: 56
```

Deploy the generated model

- Note that the tools know that this is a XGBoost container
  - Thus it knows where the model is stored
  - It uses that information to create a new container with the stored model



Send data to the deployed model

In [20]:
```
test_data_array = test_data.drop(['y_no', 'y_yes'], axis=1).values #load the data into an array
xgb_predictor.content_type = 'text/csv' # set the data type for an inference
xgb_predictor.serializer = csv_serializer # set the serializer type
predictions = xgb_predictor.predict(test_data_array).decode('utf-8') # predict!
predictions_array = np.fromstring(predictions[1:], sep=',') # and turn the prediction into an array
print(predictions_array.shape)
```

```
(12357,)
```

Two application approaches

- Build model in Spark
  1. Run model with application in Spark
  2. Export model (PMML) and run model & application in another language (Python)