CS 677: Big Data

Distributed ML

Features (1/2)

- We've already talked quite a bit about features in the context of our datasets
- But we didn't really discuss: *what are features?*
- In Machine Learning, a **feature** is a measurable property or characteristic
- Choosing the right features is the most important part of machine learning
  - **Feature Engineering**

Features (2/2)

- A feature by any other name…
  - Dimension
  - Explanatory variable
  - Independent variable
- Collections of features are called **feature vectors**
  - Or in our dataset: observations
  - These come in different types

Some Feature Types

- Numeric
- Categorical
- Boolean
- String
- Graph
- Pixels

Building Models

- For today’s discussion, we will consider two types of models
- **Regression**
  - (Prediction, Forecasting)
- **Classification**
- Many problems can be broken down into these two categories

Regression

- Statistical process for estimating the relationships between features
  - Dependent variables – to be predicted
  - Independent variables – used to make the estimation
    - Often called predictors
- Predicting income based on job history
- Estimating how much it will rain today
- Determining how long a disease outbreak will last

Classification

- On the other hand, maybe we want to **label** something based on the data
- In some cases, we may not know what the labels are
- Related: clustering
  - Taking features and splitting them into groups
- Or for example, labeling an action in a video "sitting"
- ...or whether something is a hotdog or not

Hotdog

Not Hotdog

Building a Model (1/2)

1. First, choose what you want to do: predict or classify
2. What feature (or set of features) do you want to predict?
   - These will be your dependent variables
3. Next, choose your model

Building a Model (2/2)

4. Choosing:
   - https://spark.apache.org/docs/latest/ml-classification-regression.html
   - Often best to start with something simple
   - Linear regression?
     - Not “cool” but surprisingly powerful
5. **Train** your model
6. And finally, **evaluate** your model

Training the Models

- Let's assume we've already decided on a model
- Now we need to feed it with some data so it can learn
  - **Training**
- We want to create a model that will generalize to new, unseen data points
  - But usually these new data points don't exist yet (or we don't have them)
- So instead, we **partition** our existing dataset…

Dataset Partitioning

- First step: **shuffle it!**
- Split our dataset into two parts:
    - Training dataset
    - Test dataset
- A 70/30% split is good, 90/10% is common
    - We want to train our model using the most data possible, but we also want to be able to evaluate it well
- **Never ever** use your entire dataset to train and then test on a subset!!

k-Fold Cross Validation

- Another take on partitioning: break the data up into *k* folds
- Choose a fold, then:
  - k-1 remaining folds: train
  - k: test
- Repeat for each fold

Partitioning With Spark

- Lucky for us, Spark DataFrames have us covered:
- ```
  (trainingData, testData)
        = dframe.randomSplit([0.9, 0.1])
  ```
- This returns two DataFrames, partitioned and ready to

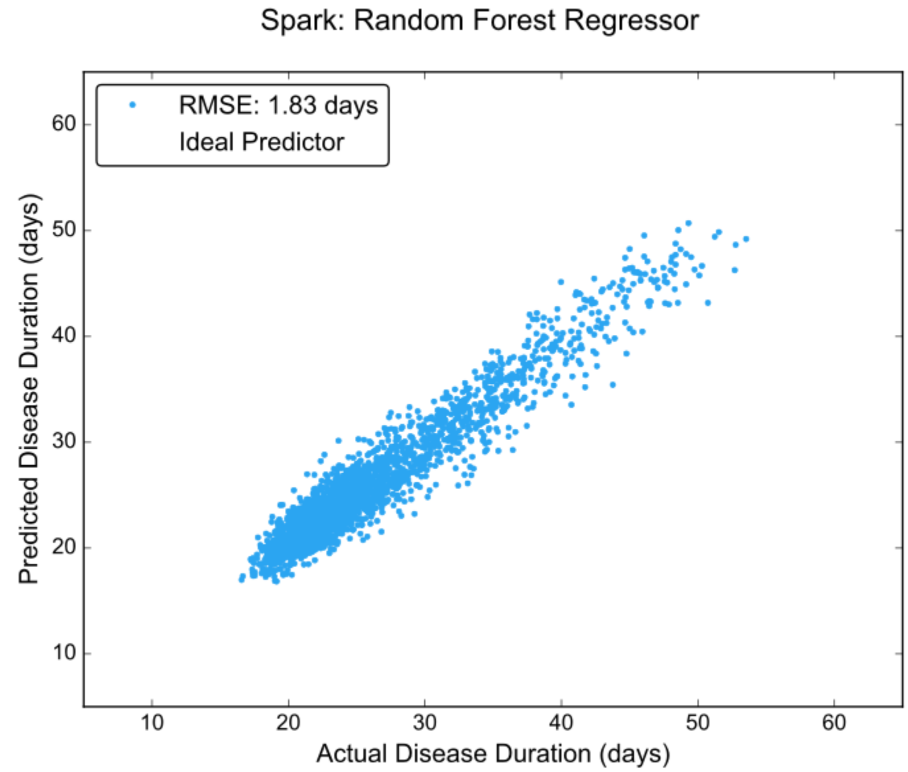
  go
  ```

Basic Model Evaluation

- We can compare **how far** predicted values are from the actual values
- MSE = mean squared error
- RMSE = root mean squared error
  - Describes the error in the same units
  - “We’re off by 2.3 days”
- Warning: this can hide issues with the model
  - Maybe things look good **on average**…

Evaluation - Regression

- Most common way to visualize accuracy: lag plot
- Plot actual vs predicted values
  - Same axes
- The closer to the line, the better



Evaluation - Classification

- For pattern recognition and binary classification, we can use **precision** and **recall**
- Precision – how useful the search results are
- Recall – how complete the results are
- So we look at true/false positives and true/false negatives

Distributed ML

- We can use Spark and other similar projects for distributed machine learning
- However, these were designed for general use; applicable for a variety of problems
- Recent years have seen the rise of distributed ML, particularly with deep learning

Parameter Server

- You just spent a semester being told that shared state is going to negatively impact performance…
- …but many (most?) ML algorithms require some level of shared state
- The *Parameter Server* design was architected for storing and synchronizing state from the ground up
  - Initially: let's used memcached to store this state!
  - Later: more optimizations, *push* and *pull* sub-model states

Parameter Server Workflow

- Use Spark/MR for:
  - Feature exploration and extraction
  - Feature engineering
- Ingest this data into a parameter server
- Apply operations on ***tensors***
  - Multidimensional array
    - (Note: this is a bit of a simplification)
- Profit

Frameworks

- Two ways to go with distributed ML now:
  - TensorFlow
  - PyTorch
  - (Epic Google vs Facebook battle)
- Both of these can exploit GPUs, have easy-ish frontends, lots of built-in models and functionality

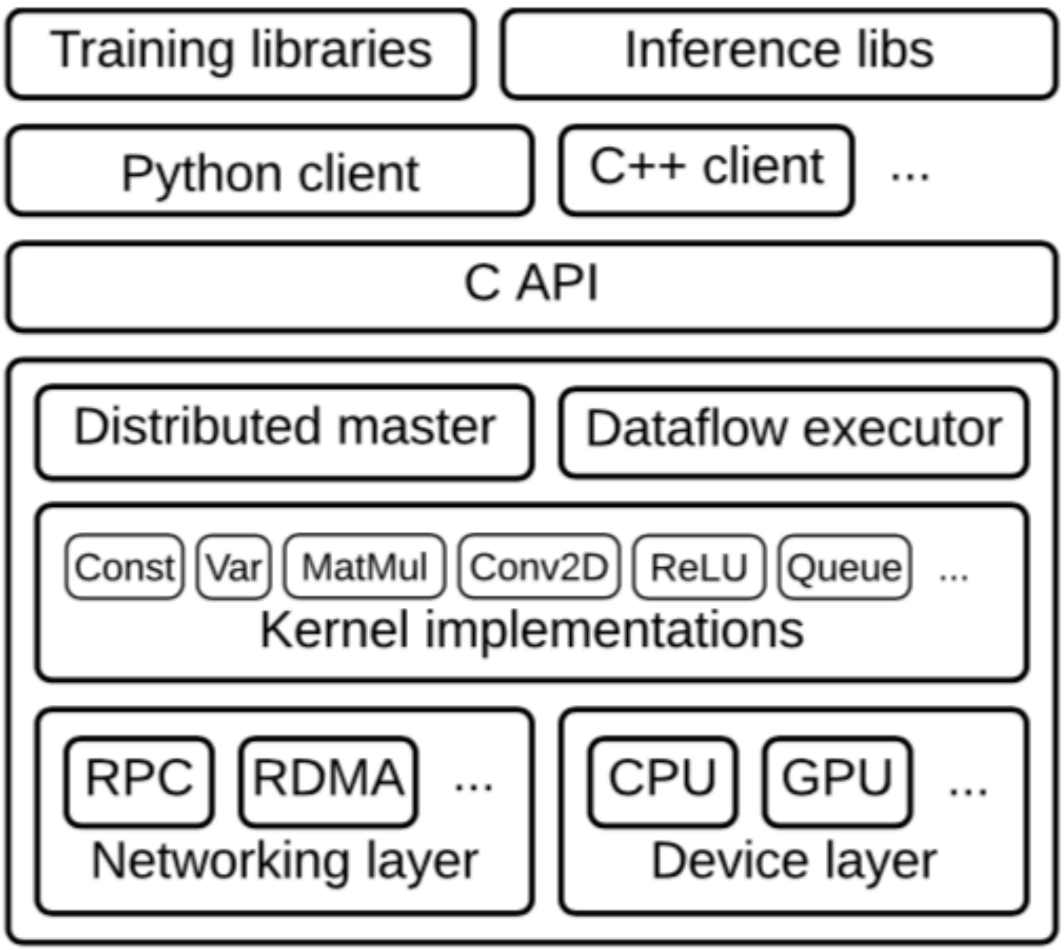
TPUs

- You may have heard of Google's Tensor Processing Units (TPUs)
  - What are these things?
  - ASICs – application-specific integrated circuits
    - Think along the lines of specialized crypto mining hardware
- Basically, a specialized hardware component for doing lots of low-precision calculations

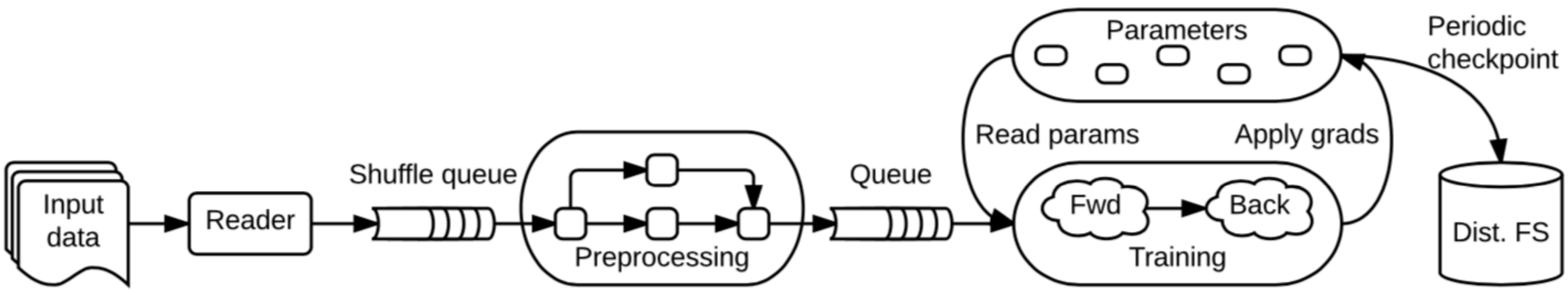
TensorFlow

- We have **tensors** of our data
- Each **operator** takes a tensor as input and produces a tensor as output
- Tensors organized in a dataflow graph
  - Vertex: operator
  - Edge: tensor
- See: https://www.tensorflow.org/about/bib

Architecture



Dataflow Graph



Feature Comparison

- MapReduce is a much more constrained type of dataflow graph
  - Requirement for round trips to HDFS make machine learning difficult
- Spark requires immutability and a deterministic flow of operations
- Parameter servers may be less flexible, makes assumptions about model state

Distinguishing Features

- Mutable state
- Less stringent consistency guarantees, meaning fault tolerance can be relaxed somewhat
  - Many ML algorithms are resilient to inconsistency (this was actually one of the main takeaways from DistBelief and the Parameter Server)
- Support for GPUs and Tensor Processing Units (TPUs) baked in

APIs

- Using TensorFlow is quite nice – Python API
- Extending TensorFlow can be done in C++
  - Well-known by developers
  - High performance
- Each operator can have multiple implementations depending on hardware
  - Run on CPU, GPU, TPU, etc
  - Great for dealing with heterogeneous clusters

Spark vs Tensorflow

- TensorFlow is probably better suited for ML applications than Spark, Flink, etc.
  - It's designed for ML rather than general distributed computation
- We have to deal with a trade-off here: is integrating our feature extraction/exploration/engineering with model building important?
- We can also combine these…

Spark + TensorFlow

- There are a few examples of using Spark to parallelize individual TensorFlow jobs
  - For instance, searching for the best parameters: Spark runs multiple TF instances
- You could potentially pass data to TF from the Hadoop Reduce phase (?)

TensorFlow Lite

- TF Lite can be used to deploy models to mobile devices, IoT nodes, fog nodes, etc
- Training is done on a powerful cluster of machines
- The trained model is transferred to the device and used to predict/classify/etc.