Machine learning at scale

- Combining A, M and P in a real application:
  - Complex models (car traffic estimation)
  - Crowd-sourced data (mobile phones)
  - Computations on the cloud
- How we run Spark inside *Mobile Millennium*
Plan

- Why *car* traffic estimation
- Overview of *Mobile Millennium*
- 2 minutes of applied Machine Learning
- Programming with the Spark framework
- Conclusion: the good, the bad, the not so beautiful
Need for good traffic estimation

- Traffic congestion affects everyone
- Up-to-date estimation is critical
- Complex for urban streets (*arterial roads*)
Real-time processing of fleet data

- Input: sampled position of taxicabs
- Observed every minute
Estimating the travel times

- Input: sampled position of taxicabs
- Observed every minute
- Covers the whole SF Bay
- 0.5 Million points / day (60M / day total)
- 0.1 Million road links
Filtering of fleet data

Preprocessing:
- Recovering trajectories from GPS points
Mobile Millennium

- A cyberphysical system for participatory sensing
Mobile Millennium

• A cyberphysical system for participatory sensing

Today: Batch jobs outsourced to the cloud
Estimation of arterial traffic

• Input:
  • Pieces of trajectories between GPS points

• Output: probability distributions of travel time
  • For each link
  • Parametrized by vector $\theta$ (mean and variance of link travel time)
The way things work

- Example road network

- Associated link travel times:
The way things work

\[ a \quad b \quad c \]
The way things work

Measurement sent

\[ a \quad b \quad c \]
The way things work

$T_a$

Measurement sent
The way things work

\[ T_a^{(1)} \quad T_a^{(2)} \quad T_a^{(3)} \quad \ldots \]
The way things work

\[ T_a^{(1)} \quad T_a^{(2)} \quad T_a^{(3)} \quad \ldots \]

\[ \theta_a \]
Life is not so simple

Measurement sent

\[a \quad b \quad c\]
Life is not so simple

- Long time between observations

\[ T_a + b \]

\[ T_a \, ? \quad T_b \, ? \]
Life is not so simple

- Long time between observations

\[ T_a + b \]

- Solution: sample!
Life is not so simple

• Long time between observations

\[ \begin{align*}
  t_{a_1}^{(1)}, t_{a_2}^{(2)}, t_{a_3}^{(3)}, \cdots \quad & \text{and} \\
  t_{b_1}^{(1)}, t_{b_2}^{(2)}, t_{b_3}^{(3)}, \cdots 
\end{align*} \]
Life is not so simple

- Long time between observations
Life is not so simple

- Long time between observations

\[ T_{a+b}^{(1)}, T_{a+b}^{(2)}, \ldots \]

\( t_{a(1,1)}, t_{a(1,2)}, t_{a(1,3)}, \ldots, t_{a(2,1)}^{(3)}, \ldots \)

\[ \theta_a \]

\( t_{b(1,1)}, t_{b(1,2)}, t_{b(1,3)}, \ldots, t_{b(2,3)}^{(2,3)}, \ldots \)

\[ \theta_b \]
Machine learning without saying it

- Procedure called *Expectation Maximization*
- Iterative in nature:
  - Alternates between sampling (E step) and learning (M step)
- Some figures:
  - 50k road links (parameters)
  - 50M observations (15GB, avg. 4 links / observation)
  - 200M partial travel times
  - x1000 samples per partial travel times
System workflow

Observations (distributed, persisted across nodes)

$\theta = \theta_a, \theta_b \ldots \theta_z$

Start link parameters (on master node)
System workflow

\[ \theta = \theta_1, \theta_2, \ldots, \theta_n \]

Network parameters (distributed over the nodes)
System workflow

For each observation link

\[ \theta = \theta_a, \theta_b, \ldots, \theta_z \]
System workflow

Travel time samples aggregated on a link basis

\[ \theta = \theta_1, \theta_2, \ldots, \theta_n \]
System workflow

New parameters are generated
The maximize sampled travel times for each link.

The master collects the vector of new parameters.
Using the Spark programming model

Main loop of the program

```scala
val observations = spark.textFile("hdfs:...")
  .map(parseObservation _)
  .cache()

var params = // Initialize models parameters
while (!converged) {

  val samples = observations.flatMap(obs =>
    generateSamples(obs, params))

  params = samples.groupByKey(false).map(
    case (linkId, vals) =>
      mostLikelyParam(linkId, vals)
  ).collect()
}
```

Step 1 (E step)

Step 2 (M step)
The good

• Before using Spark:
  • 3.5x *slower* than real-time
  • Could not even handle all the data

• With Spark:
  • Similar programming interface (methods on scala collections)
  • Very good scalability (near linear)
  • Each iteration 3x faster than reloading from disk

NERSC cluster: quad-core Xeon 4X QDR InfiniBand interconnect
Efficient utilization of memory

- The observation data is stored in memory:
  - Be careful with the memory footprint
  - Look at logs to monitor GC status

- We cache pointer-based structures
  - Significant overhead in the JVM

- Workaround: use compact collection structures (arrays) and make liberal use of `.toArray()`

- Workaround: RDDs of serialized data
Broadcast of large parameters

- Need to share data between all workers:
  - At the start of the job (network description, > 40MB)
  - Between iterations (updated parameters \( \theta \))
- Using Spark's broadcast
- Data loading time reduced by 79%

```scala
val network = // load network
val observations = spark.textFile("...")
  .map(parseObservation(_, network))

val network = // load network
val bc_net = spark.broadcast(network)
val observations = spark.textFile("...")
  .map(parseObservation(_, bc_net.get()))
```
Conclusion

• An application of Spark:
  • Real-world ML problem
  • Crowd-sourced data
• Implementation now (much) faster than real time
• Not limited by computations:
  • We can use more complex ML tools than before
Thank you