Zocdoc

Hassle free ETL with PySpark

July, 2016

- Me
- What do we want?
- Learn by doing
- ETL principles

whoami?

- Rob
- Data Scientist at Zocdoc*
- Corgi enthusiast

* Pssssst: we're hiring!

Machine Learning Engineer

www.zocdoc.com/careers

This is Ellie. She's a corgi.

- Me
- What do we want?
- Learn by doing
- ETL principles

We want *lots* of stuff

1. *Ease of access*: the ability to explore data in an ad hoc way
2. *Reproducibility*: package a job that automates our ad hoc insights
3. *Scalability*: use the same code when our data is 100x
4. *Reusability*: **huge** bonus points if we can reduce code bloat

This is *Hassle free ETL with PySpark* for a reason

- *Ease of access* and *Scalability* are super easy.
- Simple three step process
    1. Choose one of Databricks, Jupyter or Zeppelin
    2. Use your selection on a Spark cluster.
    3. No more steps.
- I use Databricks. I'm not a rep, I just really like it.

Ease of access?

- We’re Python people.
- We thought the iPython notebook was cool before it changed its name.
- We can do ad hoc analysis in notebooks. It’s fun.
- Databricks/Jupyter/Zeppelin are all notebooks!
- This is a no brainer.

Scalable too?

- Big time.
- Spark is a distributed, in memory computational engine
  - It’s super fast, easy to write and awesome
- Core Spark is written in Scala, but there is a wrapper, PySpark. Yay.
- Have more data?
  - Just use the same code on a bigger cluster

- Me
- What do we want?
- Learn by doing
- ETL principles

First, what is *ETL*?

Extract	`raw_dat = sc.textFile('somewhere') \ .map(json.loads)`
Transform	`clean = raw_dat.map(transform_one) \ .map(transform_two) \ # ...`
Load	`save_somewhere_nice(clean)`

ETL jobs can accumulate

- It starts slow: take one messy source and turn it into one clean table
- Someone really likes that datawarehouse-y table
- They want another … and another
- The summary stats are so good and the graphs so pretty it goes on.
- Now you have jobs running all night

Observe: There are more Ts and Ls than Es

- This means that we have the opportunity to share extracted data
- Recall that we're using Spark, an in memory engine
- We can construct dependency trees that, for a given E
  1. keep data in memory within our Spark cluster
  2. share it with other jobs that use the same source
  3. destroy it once all immediately dependent jobs are complete

Relatively easy way to manage this tree …

- … a super small Python package, **treetl** (find it on GitHub)
  - Tree + ETL = treetl. Get it?
- The job runner in **treetl** will
  1. maintain tree dependencies (job order)
  2. cache data when needed
  3. pass a job’s transformed data along to the next (should it want it)
- FYI This isn’t a web app/hosted scheduler, no GUI, et c. Soon? Nah.

Finally, learning by doing

```
# pull some ugly raw data from S3 just once. Have it passed along.
class GetSomeData(Job):
    def extract(self, **kwargs):
        self.extracted_data = sqlContext.read.json('your_bucket_here')
        return self

    def transform(self, **kwargs):
        # just a pass through. treetl will make sure to cache this data
        self.transformed_data = self.extracted_data
        return self
```

Finally, learning by doing

```
@Job.dependency(some_data=GetSomeData)
class SomeJob(Job):
    def transform(self, some_data=None, **kwargs):
        # do stuff with the data passed along by GetSomeData

    def load(self, **kwargs):
        self.transformed_data.write \
            .partitionBy('some_common_sense_partition_column') \
            .parquet('the_bucket_you_want_to_save_to')
        return self
```

Finally, learning by doing

```
@Job.dependency(some_data=GetSomeData)
class SomeOtherJob(Job):
    def transform(self, some_data=None, **kwargs):
        # do stuff with the passed data. Just a transformer, no load.

@Job.dependency(sj_data=SomeJob, soj_data=SomeOtherJob)
class DiamondJob(Job):
    def transform(self, sj_data=None, soj_data=None, **kwargs):
        # do stuff with the two transformed AND CACHED sources


    def load(self, **kwargs): # save your transformed_data
```

Finally, learning by doing

- Running your job tree is easy

```
# can run notebooks in Databricks like cron jobs
JobRunner(jobs=[
    GetSomeData(), SomeJob(),
    SomeOtherJob(), DiamondJob()
]).run()
```

- We shared data in memory eliminating redundant E operations
- Let’s check off *Reproducibility* and *Reusability*

Not sold on reduction of code bloat?

- There's still some boilerplate, I know. BUT …
    - … there are data idioms to be leveraged!!!!
- Suppose data comes in daily
    - write an E(xtractor) mixin that takes latest day for a given source
    - write a L(oader) mixin that saves to a new YYYYMMDD partition
- Boom … all we need to implement is T

- Me
- What do we want?
- Learn by doing
- ETL Principles

General ETL guidelines: The Basics

- Your jobs should always be
  1. Rerunnable
  2. Tested
  3. Rerunnable
- Things blow up, errors creep into the data. That's just (data) life.
- If an error existed for one week five months ago, your job should be rerunnable for that dataset.

PySpark specific considerations

1. Save cleaned data as *parquet*. It's easy.

```
your_data_frame.write.parquet('your_destination’)
```

- If you do ETL in PySpark, you probably do your analysis there
- PySpark + Parquet = Super Fast Reads
- PS This tip also means *use dataframes*

PySpark specific considerations

2. Partition by something that is easy and makes sense for your data.
   - I like YYYYMM and YYYYMMDD. It's easy.

```
your_data_frame.write \
      .partitionBy('YYYYMM', 'YYYYMMDD') \
      .parquet('your_destination’)
```

PySpark specific considerations

3. Cache data that will be reused. It's *super* easy.

```
your_data_frame.cache()
```

- This keeps processed data in memory. Speed upgrade.
- If you use **treetl**, this will be done for you between jobs
- Within jobs, go nuts.

PySpark specific considerations

4. "Uncache" data that is no longer in use. It's easy.

```
your_data_frame.unpersist()
```

- Memory is Spark's lifeblood. Be kind.
- If you use **treetl**, this will be done for you between jobs
- Within jobs, go nuts.


Thanks.
robert.howley@zocdoc.com
@robhowley on GitHub

Z