MANIFOLD

Search

KNOWLEDGE BASE

DASK AND MACHINE LEARNING: PREPROCESSING TUTORIAL

The Python ecosystem offers many useful open source tools for data scientists and machine learning (ML) practitioners. One such tool is Anaconda's Dask. At Manifold, we have used Dask extensively to build scalable ML pipelines.

In this tutorial, we'll cover the basics of Dask, and using Dask for preprocessing. We will also provide some data snapshots and code snippets; you can access those via this public self-contained GitHub repo.

DASK IN A NUTSHELL

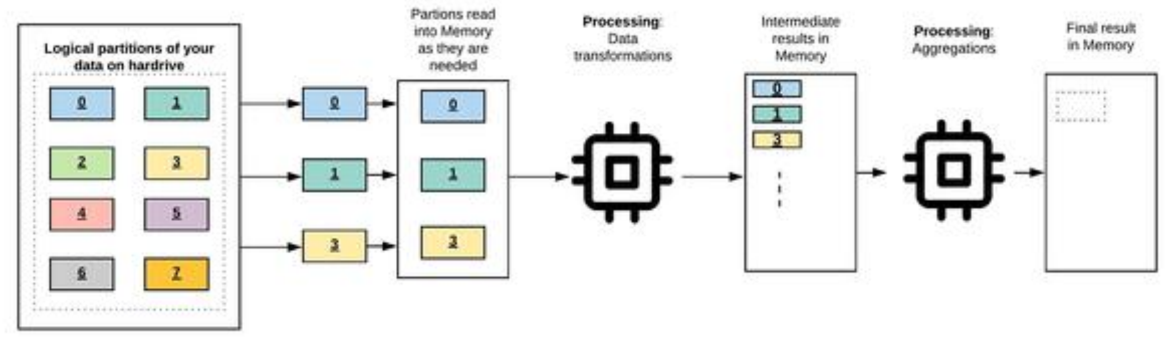
If you've ever worked with data at scale before, you know that even relatively simple operations can cause many issues. Two issues that frequently come up are:

- Your dataset does not fit in memory of a single machine.

Dask addresses both of these issues. It is an easy to use, out-of-core parallelization library that seamlessly integrates with existing NumPy and Pandas data structures.

What does it mean to do out-of-core parallel processing? First, let's look at the out-of-core part. Out-of-core processing means that data is read into memory from disk on an as-needed basis. This drastically reduces the usage of RAM while running your code, and makes it far more difficult to run into an out-of-memory error. The parallel processing part is straightforward—Dask can orchestrate parallel threads or processes for us and help speed up processing times.

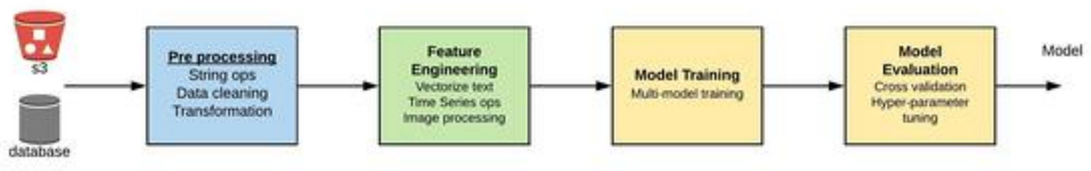
The diagram below describes this all at a high level. Let's say we want to perform an operation on our larger-than-memory dataset. If the operation can be broken down into a sequence of operations on *smaller partitions* of our data, we can achieve our goal without having to read the whole dataset into memory. Dask reads each partition as it is needed and computes the intermediate results. The intermediate results are aggregated into the final result. Depending on the specific operation, there may be many layers of intermediate results before we get our final result. Dask handles all of that sequencing internally for us. On a single machine, Dask can use threads or processors to parallelize these operations. Dask also provides a distributed scheduler that works on Kubernetes clusters.


Logical partitions of your data on hardrive
0
1
2
3
4
5
6
7
Partions read into Memory as they are needed
Processing: Data transformations
Intermediate results in Memory
Processing: Aggregations
Final result in Memory

On the parallelism front, Dask provides a few high-level constructs called Dask Bags, Dask DataFrames, and Dask Arrays. These constructs provide an easy-to-use interface to parallelize many of the typical data transformations in ML workflows. Furthermore, with Dask, we can create highly customized job execution graphs by using their extensive Python API (*e.g.*, dask.delayed) and integration with existing data structures.

DASK IN MACHINE LEARNING WORKFLOWS

The following diagram shows how Dask can be used throughout the ML workflow, from data processing to model evaluation. While there are other phases of ML, such as exploratory data analysis and error analysis, these four represent the primary workflow of a practitioner. This



DASK AND PREPROCESSING

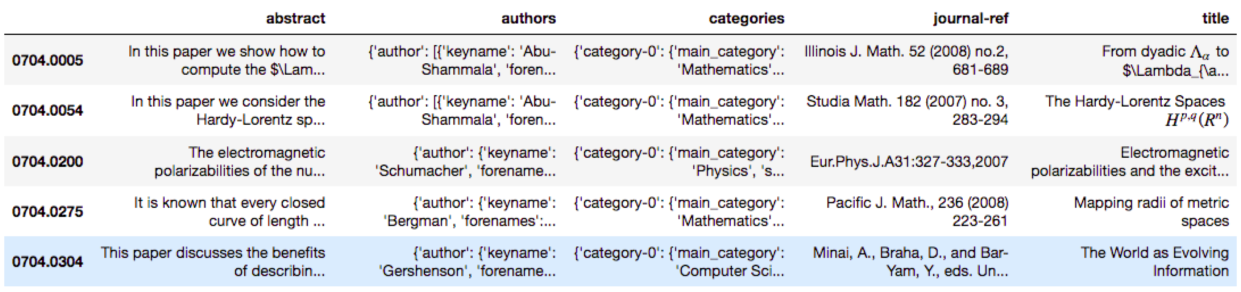
Sometimes the most time-consuming phase of dealing with data can be preprocessing, or manipulating the data into a format that you can work with for the feature engineering phase. This is particularly problematic when dealing with large datasets that do not fit in memory or have long processing times.

For the rest of this tutorial, we will preprocess the abstracts from the arXiv repository of a peer-reviewed journal article so that we can classify them into subject categories. We will describe specific design patterns and best practices to avoid out of memory errors, and use Dask to accelerate the preprocessing phase.

Although we're going to use the arXiv NLP project to convey best practices, these design patterns generalize to many other use-cases (such as image preprocessing and data cleaning).

ARXIV NLP PROJECT

As is often the case in real-world machine learning projects, our data is in a raw format, which requires some preprocessing. We have a series of JSON files, which contain information captured in the following snapshot. We will be taking a closer look at the highlighted record.



	abstract	authors	categories	journal-ref	title
0704.0005	In this paper we show how to compute the $\Lam...	{'author': [{'keyname': 'Abu-Shammala', 'foren...	{'category-0': {'main_category': 'Mathematics'...	Illinois J. Math. 52 (2008) no.2, 681-689	From dyadic Λ_α to $\Lambda_{\a...
0704.0054	In this paper we consider the Hardy-Lorentz sp...	{'author': [{'keyname': 'Abu-Shammala', 'foren...	{'category-0': {'main_category': 'Mathematics'...	Studia Math. 182 (2007) no. 3, 283-294	The Hardy-Lorentz Spaces $H^{p,q}(R^n)$
0704.0200	The electromagnetic polarizabilities of the nu...	{'author': {'keyname': 'Schumacher', 'forename...	{'category-0': {'main_category': 'Physics', 's...	Eur.Phys.J.A31:327-333,2007	Electromagnetic polarizabilities and the excit...
0704.0275	It is known that every closed curve of length ...	{'author': {'keyname': 'Bergman', 'forenames':...	{'category-0': {'main_category': 'Mathematics'...	Pacific J. Math., 236 (2008) 223-261	Mapping radii of metric spaces
0704.0304	This paper discusses the benefits of describin...	{'author': {'keyname': 'Gershenson', 'forename...	{'category-0': {'main_category': 'Computer Sci...	Minai, A., Braha, D., and Bar-Yam, Y., eds. Un...	The World as Evolving Information

We are interested in using just the abstracts to predict which category or categories the paper is classified into. So let's look at the raw abstract for this article.

```
In [7]:" ".join(raw_df.loc["0704.0304", "abstract"].split("\n"))
Out[7]:
```

```
especially in the study of the evolution of life and cognition. Tradit
studies encounter problems because it is difficult to describe life an
in terms of matter and energy, since their laws are valid only at the 
scale. However, if matter and energy, as well as life and cognition, a
in terms of information, evolution can be described consistently as in
becoming more complex.   The paper presents eight tentative laws of in
valid at multiple scales, which are generalizations of Darwinian, cybe
thermodynamic, psychological, philosophical, and complexity principles
These are further used to discuss the notions of life, cognition and t
evolution.'
```

It seems as though this paper is discussing a cross-section of a few different fields. Let's find out what category the paper is classified into by arXiv.

```
In [8]:raw_df.loc["0704.0304", "categories"]
Out[8]:
{'category-0': {'main_category': 'Computer Science',
  'sub_category': 'Information Theory'},
 'category-1': {'main_category': 'Computer Science',
  'sub_category': 'Artificial Intelligence'},
 'category-2': {'main_category': 'Mathematics',
  'sub_category': 'Information Theory'},
 'category-3': {'main_category': 'Quantitative Biology',
  'sub_category': 'Populations and Evolution'}}
```

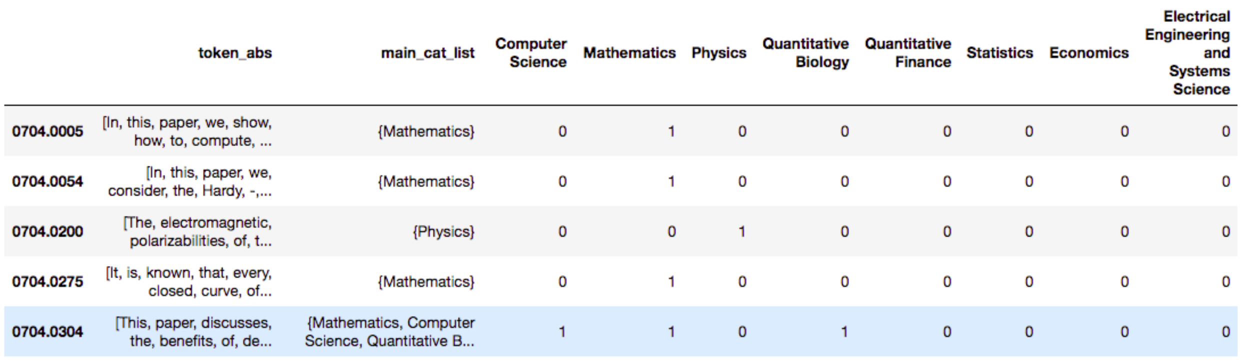
It turns out that this paper not only discusses topics from different fields, but it is even classified into three different main categories. This is what is referred to in machine learning as a multi-label classification problem. When creating our response variables, we will transform this into several binary classification problems by one-hot encoding this information and running a model for every category.

CONVERTING THE DATA

We need to convert the data from this raw form into a format we can use for feature engineering. In the context of the arXiv project, that means two separate tasks:

1. Tokenization of the abstracts
2. One-hot encoding of the categories

The results of our processing will be of the following form shown below. The token_abs



	token_abs	main_cat_list	Computer Science	Mathematics	Physics	Quantitative Biology	Quantitative Finance	Statistics	Economics	Electrical Engineering and Systems Science
0704.0005	[In, this, paper, we, show, how, to, compute, ...	{Mathematics}	0	1	0	0	0	0	0	0
0704.0054	[In, this, paper, we, consider, the, Hardy, -,...	{Mathematics}	0	1	0	0	0	0	0	0
0704.0200	[The, electromagnetic, polarizabilities, of, t...	{Physics}	0	0	1	0	0	0	0	0
0704.0275	[It, is, known, that, every, closed, curve, of...	{Mathematics}	0	1	0	0	0	0	0	0
0704.0304	[This, paper, discusses, the, benefits, of, de...	{Mathematics, Computer Science, Quantitative B...	1	1	0	1	0	0	0	0

Since our dataset is large and consists of several files, we don't want to do this in pandas on every file. We can use Dask to parallelize this workload. Imagine we have a function, `preprocess_input_json`, which preprocesses one file, and looks similar to the code below. For brevity, we can ignore the details of how the `tokenize` and `get_ohced_main_category` functions work. If you want to see the full code please check out the preprocessing notebook in our GitHub repo.

```
def preprocess_input_json(jfile, preprocessed_dir, category_list, retu
    processed_df = pd.DataFrame()

    with open(jfile, "r") as f:
        raw_df = pd.read_json(f, orient='index')

    processed_df['token_abs'] = raw_df['abstract'].map(tokenize)
    ohced_main_cat = get_ohced_main_category(raw_df, category_list)
    processed_df = pd.concat([processed_df, ohced_main_cat], axis=1)

    out_file = os.path.join(preprocessed_dir, os.path.basename(jfile))
    with open(out_file, 'w'):
        processed_df.to_json(out_file, orient='records', lines=True)

    if return_df:
        return processed_df
```

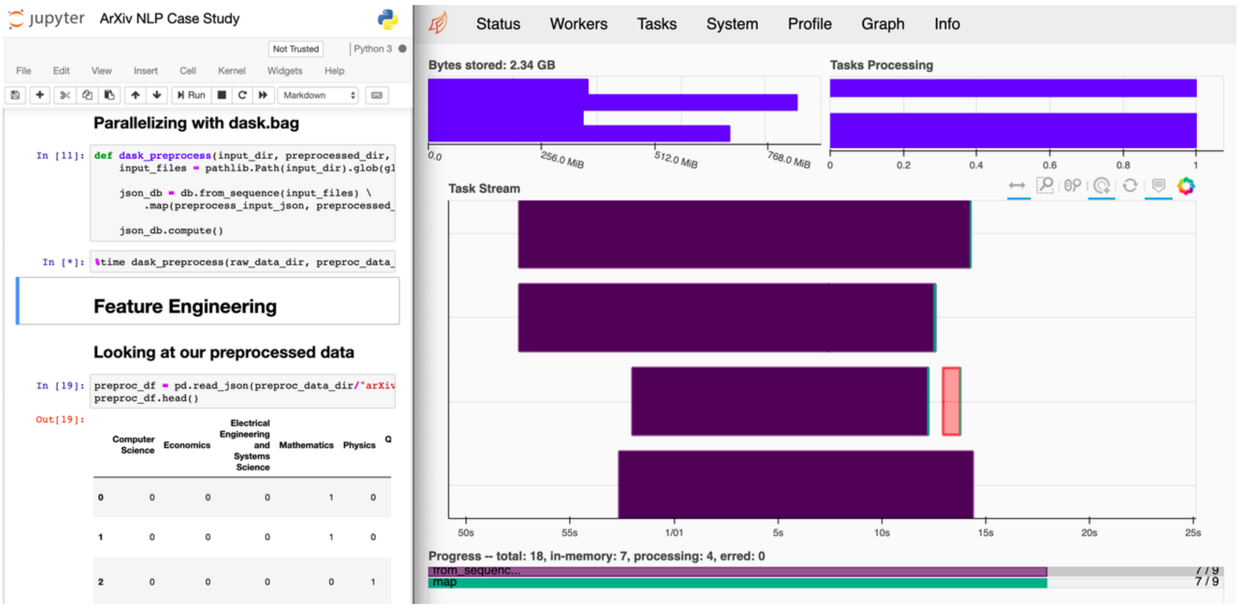
Here, we've written code to tokenize one file, but we have several files we want to preprocess in parallel. So we can use the dask.bag data structure to preprocess several files in parallel. Since we chose one JSON file to be the “unit” over which we are parallelizing, Dask's API makes it painless to write a wrapper function like the one below.

```
    input_files = pathlib.Path(input_dir).glob(glob_pattern)

    json_db = db.from_sequence(input_files) \
        .map(preprocess_input_json, preprocessed_dir, category_list)

    json_db.compute()
```

The screenshot below shows the state of our local cluster when the above function is called. On the left, we see the asterisk on the currently executing cell. On the right, we can see the dashboard. There are four workers in our cluster and they are working in parallel, each processing one JSON file. So, we have 4x parallelism compared to a serial loop using pure pandas code. If we replace the local cluster with a Dask cluster running on Kubernetes and our files on a cloud storage system like Amazon S3, we can increase this parallelism significantly to get further speedup and handle even larger datasets.



Parallelizing with dask.bag

```
In [11]: def dask_preprocess(input_dir, preprocessed_dir,
             input_files = pathlib.Path(input_dir).glob(gl

             json_db = db.from_sequence(input_files) \
                 .map(preprocess_input_json, preprocessed_

             json_db.compute()

In [*]: %time dask_preprocess(raw_data_dir, preproc_data_
```

Feature Engineering

Looking at our preprocessed data

```
In [19]: preproc_df = pd.read_json(preproc_data_dir/"arXiv
         preproc_df.head()
```

Out[19]:

	Computer Science	Economics	Electrical Engineering and Systems Science	Mathematics	Physics	Q
0	0	0	0	1	0	
1	0	0	0	1	0	
2	0	0	0	0	1	

```
Progress -- total: 18, in-memory: 7, processing: 4, erred: 0
from_sequenc...   7 / 9
map               7 / 9
```

DESIGN PATTERNS

Build incrementally

From our experience working on a variety of machine learning projects, we believe the best path to achieving an effective machine learning solution is one that starts with building an end-to-end system quickly. Once we have an end-to-end system off the ground, we can assess model performance and improve it using quantitative metrics. The quality of our final model then becomes a function of how quickly we can iterate on our approaches at each step in the ML workflow.

less time we spend knowing how well it actually works. Particularly early in the machine learning process, *e.g.*, in preprocessing, we need to get up and running as quickly as possible and be able to make tweaks and iterate rapidly.

In our case, we started with simple non-parallelized code to process one file and then scaled out by writing a Dask wrapper to execute many data parallel jobs simultaneously. Once we had simple code using pandas that worked well, we parallelized it by:

1. Putting a list of file paths into a Dask Bag
2. Mapping the preprocessing transformation function on each file path
3. Writing each transformed file back to disk in a new directory

In this way, we were able to write parallelized code in a much shorter amount of time. While one key motivation for using Dask is to reduce compute time, another important consideration is to spend our development time wisely. If we spend hours trying to get more complicated parallelized code up and running, we are inherently limiting the amount of time we can devote to iterating rapidly.

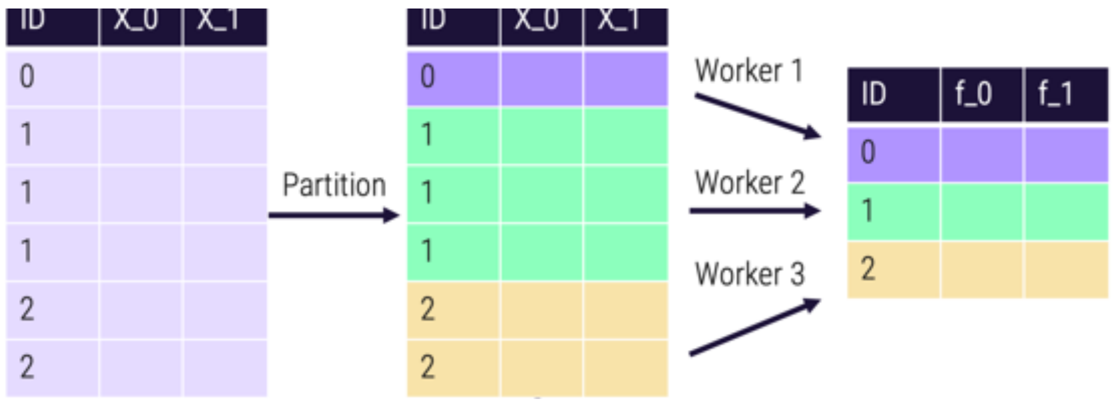
Identify the Parallelizable Unit

The implicit notion in *building incrementally* is that the code can be written such that there is a unit-parallelizable process. That is, we can write simple code to execute the process for one unit of data (*e.g.*, files, images, unique keys in a pandas dataframe, elements in a list, etc.) and then write code to parallelize this process over all units of data in the dataset.

The choice of which unit to parallelize over is an important consideration. Broadly speaking, there are two options in this type of scenario, as illustrated in the picture below. The file-level parallelism works well for this situation because:

- The overhead of spinning up worker processes is shared by processing of many records.
- We wanted to have output as separate files, one for each input file, which gives us better observability as data flows through our pipeline.

In a different situation, ID-level parallelism might be a better fit. For example, imagine a situation where we have a tabular dataset that corresponds to customers visiting a website. There may be multiple records per customer in our table. A common operation in those types of datasets is to compute some metric per customer. In this scenario, it is natural to partition the dataset by customer ID as illustrated below and our operations would be parallelized over customers. In an extreme case of this ID-level parallelism, each row can have a unique ID. In that case, our operations would be parallelized over rows.



CONCLUSION

In this tutorial we learned the basics of Dask, and how to use it for a simple preprocessing step.

Here are some other Dask resources so you can continue experimenting:

1. An excellent tutorial by Matt Rocklin, the lead developer of Dask
2. A longer tutorial with code
3. A ML Docker image we created at Manifold with all the libraries we typically rely on including Dask
4. A similar Docker image, targeted for Deep Learning with Dask, Pytorch, Tensorflow, and more

Jason Carpenter, Machine Learning Engineer

Prior to Manifold, Jason was a Neuroscience Research Associate at UCL (University College London), where he investigated specific brain regions by integrating computational modeling and neuroimaging techniques.

ABOUT

CLIENT PROFILES

OUR TEAM

CAREERS

KNOWLEDGE BASE

NEWSROOM

AMAZON WEB SERVICES