Tasks finished

Pulling Data

We explored a couple ways to pull data. We could either use the MediaWiki API or we can pull straight from the wikipedia dumps. After exploring the API for a while (which is attractive because we don't have to deal with XML), we decided that we would hit rate limiting too fast to train our model. We then decided to try and deal with the XML dumps.

XML parsing

This was somewhat difficult. We are dealing with large files (~35 gb) which means that might struggle when loading the full file into memory. Because of the way that XML is structured, it can be difficult to parse just a part of file (in contrast to csv files). A lot of tools that are built for parsing full documents of XML, but very few are built for parsing partial XML documents. In addition, writing a multithreaded parser is a lot harder than writing a single threaded parser.

We accomplished XML parsing by using spark user defined functions. This allowed us to take advantage of the same spark executors that we were using to train our model with and apply them to XML parsing as well. The way that we do this is that split the document into even chunks among all the executors that we had, and then allowed them to find instances of the tags (and their values) that we were looking for. Then, we combined all of those into a full dataset that we can then train on.

All of this only applies when testing locally. If our parser fails and is missing too much of the data, our fallback will be to just get machines with big enough memory (~64gb) and then train that way. This should not be too big of a hurdle.

Model Exploration

We looked into doing deep learning first. We ended up deciding that this was infeasible, because we would not be able to train our dataset effectively (there is just too much computation to do). Therefore we decided to use a simpler model like LDA or TFIDF. Both of these are already implemented in spark mlib, so we should not have too hard of a time implementing them.

TFIDF

This model basically figures out how important a word is inside of a document. In addition, you figure out essentially how special this word is to this document. If the term appears a lot through all the documents, then that word is not super important to this document. Similarly, if this is the only place that word shows up (and it shows up a lot) then that means that this word is very important to

this specific document. This can then be used to figure out what the document is about and then can be used to group different documents together.

LDA

This model basically comes up with groupings of words and figures out what "topics" are inside of the corpus of documents. You essentially perform unsupervised classification over the corpus and figure out some natural clusters which can then be extrapolated out into topics. This is one of the standard ways to perform topic modeling and figure out what documents may contain the same topics. Once we do this, we can then cluster the documents together and figure out the groupings.

We actually implemented and tested LDA as well on a different much smaller dataset, just to make sure that we could do the implementation.

Computation Capacity Exploration

Actual training

We will need to run a Hadoop Cluster (or at least Hadoop Yarn) in order to orchestrate the spark executors, so we explored the best ways to do this.

1. Use the NYU Hadoop cluster
2. Use DigitalOcean credits to spin up a Hadoop Cluster
3. Use a premanaged Azure Cluster.

We used to like choice #2, however we are realizing that we will not have enough credits to train our model fully and are realizing that we will need to seek access to the NYU Hadoop Cluster.

Development Environment

We realized that dealing with these massive files locally is becoming a bit hinderance and we are spending quite a bit of time with memory issues. We think that we might have to move development off our local machines onto the HPC JupyterHub instance, which we will also need to seek access for.

Action Items

1. Get access to HPC
2. Get access to Hadoop Cluster
3. Acutally parse all XML in dumps and put into a DB
4. Run LDA on a subset of that DB to decide Hyperparameters
5. Run LDA on the full DB and get results.