Here are some general approaches you might try for unsupervised learning on text. There are no good rules regarding what tools work best – it really depends on the data you have to work with and the types of insights you’re looking for.

1. **Text Extraction**: We’ve tried a variety of tools to convert documents like html, xml, and pdf to plain text, but the most helpful have been:
   1. BeautifulSoup (<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>) works very well for html and xml. There are several parsers you can choose from, including Python’s built-in html parser and lxml. I usually start with lxml but will try one of the others if results are poor.
   2. Pdftotext: this is a linux util that comes pre-installed on most builds (if not, you should be able to apt-get or yum it). We usually just run a short bash script that reads pdfs from one directory and dumps .txt files to another. Results really depend on the quality of the source .pdfs. It seems to provide well-formed text about 80% of the time. Another 10% of docs will yield words and phrases sufficient for tf-idf and other statistical methods.
   3. AWS Textract is an OCR service that can be helpful for .pdfs that are image scans of text, and other images with text in them.
2. **Cleaning**: depending on the quality of the source documents, you may need to do some light or heavy cleaning, and potentially dealing with encoding issues. I haven’t found a good substitute for regex yet, but ftfy (<https://ftfy.readthedocs.io/en/latest/>) will help with a lot of common Unicode issues.
3. **Word/phrase extraction**: we get the best results from using noun phrases. There are a few general approaches here (certainly more than that – this is just what has worked for us):
   1. Extract noun phrases using a tool like Spacy (<https://spacy.io/>), Gensim (<https://radimrehurek.com/gensim/>), or NLTK (<https://www.nltk.org/>). I like Spacy for its flexibility, but sometimes it might be a bit slower then Gensim or NLTK. I’d recommend trying them all to see what works best for you.
   2. Extract keyphrases using one of the dozens of models that have come out in the past decade or so. There’s a fantastic overview in (<https://arxiv.org/abs/1905.05044>), and you can access most of those models either through pke (<https://github.com/boudinfl/pke>) or textacy (<https://github.com/chartbeat-labs/textacy>). In general, we’ve found that some of the graph-based approaches (TextRank and derivative models) yield disappointing results, but they’re worth trying as a baseline. TopicRank and Topical PageRank seem to generate better-differentiated keyphrases, but they can be slow, and Topical Pagerank requires training an LDA model over the corpus that the model uses to extract document-level phrases.
   3. Tokenize and use unigrams (or bigrams etc.) – simplest approach and sacrifices the context offered by noun phrases, but sometimes serves as a good baseline, and can work well with really messy text where phrases are difficult to parse
4. **Clustering**: This piece is probably the most challenging but the most rewarding. Here are a few approaches that have worked for us:
   1. Phrase counts >> tf-idf >> SVD (i.e. LSI). We generally prune the counts to a minimum occurrence across the corpus (e.g. omit the phrase from vocab if it occurs in fewer than 5 docs). If you’re not familiar with SVD/LSI, there’s good coverage in (<http://www.mmds.org/>), (<https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf> – chapter 18), and (<https://web.stanford.edu/~hastie/ElemStatLearn/> - chapter 14) SVD is implemented in scikit-learn and also TensorFlow
   2. LDA (<http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf>) – Bayesian model that factors word/phrase counts as discrete draws against a topic, where documents represent a multinomial distribution of topics. There’s an implementation in scikit-learn. LDA can be challenging to implement because it’s very sensitive to priors (mostly the number of topics), but it can be a great place to start identifying topics related to human capital and human rights
   3. Hybrid approaches employing pre-trained embeddings. These have been some of the most effective given the power of embeddings. The Flair library (<https://github.com/flairNLP/flair>) has a great collection of embeddings from GloVe and FastText to the embeddings from transformer-based models like BERT, GPT2, etc. One approach you can try is representing documents by embedding the top k keyphrases or noun phrases, then weighting those embeddings by their keyphrase score (from pke for example) or tf-idf score. We then typically use some sort of dimensionality reduction on the embeddings (UMAP is really helpful) to do clustering and/or viz.
5. **Viz**. Not sure if this is something you want to focus on, but we’ve relied mostly on d3.js, and have been using scatter-gl (<https://github.com/flairNLP/flair>) to do viz on document sets. Alternatively, matplotlib and seaborn are easy options. We can demo some of this for you if you’d like.