Vimeo staff picks challenge writeup

A one page writeup on my experience with this coding challenge.

Your approach used and what you liked and disliked about this challenge.

The first problem I encountered when solving this challenge was deciding how to measure the similarity of clips. I was given several features to chose from, but no ground truth to help solve this problem, so I had to chose an unsupervised model. I chose only to use the clip title, caption, and categories as features, but I will touch on using more features in my improvements section. Now that I had decided what features I would use, I had to decide how I would make my predictions. The two suggested approaches, creating an inverted index, or clustering the results of word embedding and performing a nearest-neighbors search, seem to be the best approaches. Clustering allows for deep exploration of the data, but can be a costly solution. I decided training a useful model for word embedding and then finding nearest neighbors with a set of clips which is frequently updated would be hard to scale. So I settled on using an inverted index with TF-IDF information to find similar clips. Elasticsearch provides a robust inverse index, and comes with a built in similarity measure which uses TF-IDF, so I decided to use their existing implementation. I chose to weigh category, title, and caption equally when calculating clip similarity.

After settling on the back end, I considered my front end approach. I wanted to not only provide a command line interface for interacting with my solution, but also a web app. I settled on Flask to build web app, and because I now had several dependencies I Dockerized the full project to enable deployment using docker-compose.

An additional design decision I made was to use elasticsearch-dsl's persistence which creates a persistent layer in my ES index, allowing me to access the index in a Pythonic manner.

General description of the project:
main.py - command line interface, can return similar clips given clip ID, index new data, and more.
run-flask.py - web interface, returns similar clips given clip ID.
vimeo_challenge - Python library for the challenge. - data.py - Loading functions for data, has data_to_index method which abstracts the indexing of the data. Also used for printing of json output in main.py - index.py - Class for persistence, methods to create ES index (if necessary), fill index, restore snapshot of index, save snapshot, return similar clips given ID. - tests - unit tests for index.py and data.py

I really enjoyed working on this challenge. Learning about modern word embedding techniques when researching my approach was very interesting. I'm a big fan of working fun projects that let me explore topics I haven't worked with much, and with this project I tackled issues I'd never dealt with, and learned a lot in the process. Now I want to see what this problem looks like at full scale at Vimeo!

How would you go about building this for real at Vimeo?

There are three main sections of the project I'd address if building this for real: a robust similarity measure, a scalable implementation, and a well designed front end. The first thing I'd do is explore data I did not have access to when completing this challenge. There are probably better predictors for clip similarity than title, and I may be able to build a model significantly more robust than just using ES's built in "more like this" function. It's worth noting that you can implement custom similarly measures for ES. When it comes to building a scalable implementation, Elasticsearch is actually a good choice (if it also solves the problem). I don't know exactly how much data would be handled, but with careful planning very large ES indices can be searched very rapidly, and if new indices are regularly created as new clips are posted to Vimeo, re-indexing becomes a more manageable problem. Finally, the web app I built has a lot of room for improvement. There are several metrics that could be displayed showing users how the similar clips were selected which I would like to include.

Are there additional techniques you would leverage to improve the results?

The obvious things missing from my implementation which would improve results are: - Removal of more stop words (for example, names cause issues in my current implementation) - Language is an issue. Decide how to deal with clips in different languages (this issue may not be as serious with more clips). - Weighting features. By setting a parameter in the ES query I can specify how important caption should be to the metric. This could be dynamically changed so that if, for example, there is no caption, there will be no weight allocated to the caption in the similarity metric so that we are not only returned other clips with no caption.

If performance is not a consideration, t-SNE (clustering) applied to documents seems to be a promising approach, as described in the Scikit-learn documentation.

What performance considerations are important to scale this up to all Vimeo videos and why?

A big performance consideration is the cost of indexing data. Having to re-index large quantities of data can put a search engine out of service if the cluster is not well structured. Maybe creating new indices once per day or week could counteract these issues. Deletion from indices can also cause problems if done in large volume, so if there is frequent removal of clips this is something that would need to be addressed.

The number of queries to the index is important as well. If we expect a high volume of queries using a similarity metric which is quickly calculated is important. In addition, the client facing server will need to handle heavy traffic. Relevant ES documentation.

Finally, it may be beneficial to use several ES clusters rather than one large cluster -- source.