Prognozirovat - Binary Classification

Dharmesh Tarapore
weirdindiankid
dharmesh@bu.edu

June 29, 2017

Problem

I was given a training dataset consisting of a YouTube video id followed by its length, age, the uploader's YouTube username, the video's category, the video's aggregate rating on a scale of 1 to 5, the number of ratings, the number of comments on the video, and up to 20 ids of recommended videos.
Given this data, I was expected to predict the existence of an edge between two video ids (source, target) in the testing dataset. Using a shortest path algorithm across the entire data represented as a weighted graph, I was able to achieve an accuracy of about 93%.

Initial Approach and Considerations

A difficulty arose when I found that several *target* video ids were nowhere to be found in any of the supplied data files. While I initially considered leveraging YouTube's API to scrape the data I would need to make accurate predictions, I was unsure if that would be against the rules. I was also not sure those videos would be available on YouTube anymore.
Having abandoned this approach, then, I decided to predict the non-existence of an edge for *targets* that could not be found anywhere else in the dataset.

Preprocessing

Using pandas, I was able to visualise the data to gain an understanding of how it was structured and identify the most important features. This proved more difficult than I had anticipated. I was quickly inundated with way too much information. Unsure

about which book to read, and short on time, I decided to start with the most obvious approach: work to visualise the data as a graph first. The handout mentioned the data was structured as a directed graph, so I chose to go ahead with this approach. Every distinct video id is represented as a node, while every recommended video for a given video is another node connected by an edge of weight 1.
Then, using *Djikstra's* algorithm, we can find the shortest path from a source to its target for every video in the test set. If a path did not exist, we assigned it a ridiculously high value that we could then use to assert the non-existence of an edge when building the final submission data file.

Issues with the Intermediate Step

Representing the data as a graph took about 3 minutes on my MacBook, but calculating the shortest path between neighbours was truly daunting. On 2 separate attempts, the kernel killed the python process to save memory. I decided to attempt to optimise my function by using the *blackbox* module for Python, written by Paul Knysh and Yannis Korkolis. This proved futile as well. I thus decided to run my code on my personal AWS EC2 instance, with 256GB of memory. After about 15 hours, I had a csv file containing the shortest path between each *source* and *target* video in the test dataset. Now all I had to do was calculate a threshold below which I could definitively say an edge existed. Looking at the measures of central tendency for the list of scores minus those with the ridiculously high score previously assigned (I did not want these values skewing the metrics), I decided to use a threshold of 4. That is, if the length of the shortest path between two videos is greater than 4.0, we say that no edge exists between them. I wrote this out to the CSV file and submitted it to evaluate my results.

Other Approaches Considered

I had also considered calculating the personalised PageRank scores for each *(source, target)* pair. Inspired by Edwin Chen's blog post on how he predicted the existence of missing edges in a graph, I proceeded to code this solution. Unfortunately, an indiscernible combinatorial explosion appeared to keep happening and even my high powered AWS EC2 instance was unable to run this program. Networkx's library function did not help either.

Conclusion and Future Considerations

1. I could not figure out how to represent the data as a directed graph. I imagine this will improve the accuracy of predictions but I have no data to back this claim up (yet).

2. In assigning *edge_prediction* values to pairs where I could not find the *target* anywhere, I think a better approach might be to assign the values along a probabilistic distribution (such as a uniform or a normal distribution).

3. The edges are currently weighed equally. This is certainly not the case with the actual video recommendations, since the 20^{th} recommendation is likely to be farther from the source than the 1^{st} recommendation. I plan on weighing the edges differently in future projects.

4. I considered using video metadata and other features to build a better classifier. The slow nature of processing keeps me from being able to pursue that goal at this time.

5. A description of the files in the repository can be found in the README.