Analysis of Professional Cycling Results as a Predictor for Future Success

Alex Bertrand

Introduction

As a competitive sport, road cycling lies in a category nearly all its own. Putting aside the sheer length of the races (100 miles is, by most accounts, a short day) and the wildly varying natures of the courses themselves (mountains, cobblestones, and tight city corners are all part of the fun), it is unique in that is a particularly elegant blend of group and individual dynamics. On one hand, it is a fiercely individual sport. Team awards are given, but at the end of the day there will only be one man in each spot on the podium. And yet on the other, each and every rider is at the mercy of the group. A professional peloton is a constantly evolving blend of different skillsets and goals, constantly changing in response to moves made by individuals. It is because of this sort of interaction that I am interested in examining cycling from a network analysis perspective.

Problem

In this project I will attempt to use the results of the spring cycling season to predict the finishing order of the biggest and best cycling race in existence, the Tour de France. Of course, predicting it exactly is going to be nearly impossible, so my goal for this work will simply be to come as close as possible.

For my data I will be using the results of four of the five 'Monuments' of cycling, as well as two stage races. Each takes place at a different point in the spring, and each is well attended by some of the top names in cycling. They were chosen as such to provide maximum overlap with the names that compete in the Tour.

The six races are as follows:
Milan-San Remo
Tour of Flanders
Liège-Bastogne-Liège
Paris-Roubaix
Tirreno Adriatico
Criterium du Dauphine

Related Work

Works performing any sort of meta-analysis of cycling have been few and far between; one would go so far as to say largely nonexistent. Large amounts of energy have been put into studying the physical science behind cycling but comparatively little has been done in the 'Moneyball' vein.

Network analysis has been applied infrequently to other sports, but in methodologies which I have found to be largely irrelevant to the problem I'll discuss here. Studies of passing dynamics in basketball, for example, have little to no bearing on the predictive sort of analysis I seek to complete in this project.

Model/Algorithm

Note: Of the three approaches I tried, the latter two used the same model for the data; as such, this is what I'll describe in detail. The first approach was mostly necessary in terms of 'getting to know' the data, but less significant in terms of results. The code for it is still present in my submission, but I'll skip over it here.

I chose to model the data as a directed graph comprising riders (nodes) and performances (edges). In other words, each rider was represented by a node and each race represented by an edge between one rider and another. A single edge corresponds to the source node coming directly behind the destination node in the race – if there is an edge from A to B, for example, it would mean that A got place N in a race and B got place N-1 (lower is better). There is a natural directedness to race results in that there is a (somewhat) clearly defined order of superiority in the finish; drawing edges between adjacent finishers conveys this flow of talent/speed.

Following outward edges, then, corresponds to moving towards faster and faster riders. The hope for this model was that such a flow could be used to correlate riders with their relative speed. In practice this proved difficult; this will be discussed more shortly.

Data Scraping

All data used for this project was obtained via web-scraping or copying from cyclingnews.com. This was particularly convenient in that all the data used was available from a single source, and in a consistent format.

This made working with the data relatively easy, with the small exception that occasionally names of riders would be slightly different, most often around special characters (eg Florian Senechal vs. Florian Sénéchal). It is necessary to maintain consistency across riders to properly correlate one of their results with the other, so

there is a section of the code that deals specifically with finding the nearest matching name in the results data at runtime.

Evaluation Metric

In order to gauge the efficacy of a particular algorithm it is necessary to have a metric by which we can evaluate its accuracy. It is goal to predict Tour de France results given the results of the earlier season reasons. As such, we need some way of measuring the finishing order an algorithm outputs against the actual finishing order (of the 2017 Tour).

For these purposes, and after also evaluating the Kendall Tau and Spearman Rho, I settled on a metric based on set intersection for reasons of simplicity and focus. This particular calculation, as we'll see, effectively weights the higher ranks more while maintaining a relatively simple implementation. This former in particular is desirable for this effort; it is much more important to predict the top finishers in the Tour than it is to get the exact order of the masses right.

The 'set intersection score' (because if there's a better name, I don't know it) is computed as follows:

1. For i=1, Compare the first i elements of each list (in this case, our predicted finishing order and the actual results). Treat each as an unordered set, and calculate the fraction of elements that are shared between the two, ie
$$\frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$
This is the 'score' for that value of i.
2. Repeat step 1 for all i's up to the size of the smaller of the two lists, getting a score for each step along the way.
3. Average all scores to get the final set intersection score.

Construction of the actual graph was a straightforward process of going through the result set for each of the 6 races and adding an edge between consecutive finishers as determined by time. More than one edge between any two given riders was both uncommon and disallowed, largely in keeping with the constraints imposed by the snap.py graph representation. Two edges between any two given riders (one directed edge in each direction), however, was fine. This scenario arose when the first beat the second in one race, but lost to him in another. A looser version of this case, in fact, is one of the larger issues I believe I faced; cycles in the graph made reasoning about which riders were 'strictly better' than the others quite difficult.

The algorithm performed on this data went through two main iterations:

Breadth First Search

Given the notion that an edge pointing towards a node indicates a rider for which that given node is superior, one might logically surmise that the more nodes point (either directly or by proxy/flow) to a given rider, the faster that rider is. As such, I evaluated a scoring system under which a rider is assigned a number of points equal to the number of nodes that can be reached with a BFS search starting at their node, and following incoming edges, outward. In other words; a rider's speed under this model is directly proportional to the number of nodes that are connected to him (note that it is not the number of nodes he is connected to – subtle distinction). Riders can then be ranked by the number of points they accumulate in their respective searches.

Implementing this algorithm without any modification, however, yields an interesting result: nearly every rider has the exact same score. This will be discussed in more details in the 'Challenges' section of this report, but for now suffice to say it is due to the variance present in racing and presence of cycles in the graph.

In the interest of solving this problem I chose to implement a depth-limited BFS – essentially, only allowing progress to continue for a certain number of levels of search. Empirically, this proved effective in differentiating score among riders. This iteration of the algorithm performed somewhat ineffectively, peaking at a set intersection score of ~/.186.

In initial implementations of the algorithm I incremented a rider's score by 1 for each node their search touched. I propose, however, that this favors slower riders in that it weights the (slower) riders they beat just as highly as the (relatively faster) riders that the top athletes manage to best. To remedy this I changed the increment value to be dependent on the current depth of search at which a node was reached – essentially, placing a high value on beating riders that in turn beat many other riders or, put another way, having a high fanout. I tried both a linear and squared proportionality, and squared (that is, incrementing a riders score by $[\text{depth}]^2$ for each node that is touched during the search) performed slightly better. A depth of 10 with this i^2 increment yielded a set intersection score of ~.297.

Random Walks

In an attempt to remedy the cycle problems that BFS search faced, I implemented a ranking system based loosely on the random walks we have seen for PageRank. Starting from a random node, the algorithm followed outward links (this time moving up the speed gradient) for a specified number of repetitions or until it hit a dead end. Each rider it hit on the walk had their 'score' increased, and the riders with the highest scores at the end were taken to be the fastest.

Inasmuch as the edges (in theory) only move from slower to faster riders, repeatedly moving along those directions should narrow to faster and faster riders and, with enough repetitions, emphasize those riders at the front of the peloton.

In similar fashion to the BFS, it proved to be useful to weight later nodes on the random walk more, this time because there was a higher probability that they were towards the faster end of the group. An i^2 proportionality again outperformed its linear counterpart.

This approach was moderately effective, reaching a set intersection score of ~.240.

Challenges

Far and away the biggest challenge was the messiness of the results data. By this I mean the lack of consistency; a rider might finish in 2nd one day, and then 34th the next. This is more or less normal for cycling ('bunch sprints' at the end of the race, as they are called, oftentimes mean that such the difference between 1st and 30th might only be three or four seconds) but it makes it very difficult to get an accurate feel for who exactly is 'faster' than whom.

Add to this the fact that racing, as a human athletic endeavor, is naturally inconsistent and it becomes quite difficult to get an accurate picture of the standings, at least in a computational sense. These two factors in particular combine to generate a graph with many cycles and irregularities that makes generating continuous speed gradient nearly impossible. Approaching this with random walks was an attempt to deal with that sort of randomness, but it ultimately proved a little too difficult to surmount simply.

Results and Findings

Depth-limited breadth first search with a squared proportionality for score update performed best of the implemented algorithms with a set intersection score of ~.297.

Future work

Given more time, I would have like to spend time investigating alternate methods of representing the data in graph form. I think what I eventually came upon was adequate, but my suspicion is that an even slightly altered representation (perhaps something more adept at dealing with cycles) would greatly increase effectiveness on the algorithm design side.

One think I would have liked to have tried when parsing the data was binning results by finish group instead of finish time/order. As stated, cycling races often end with large groups of riders finishing at nearly the exact same time – so close, in

fact, that races will most often list long groups as finishing at the same time so long as there is never more than a few seconds gap between them. The data representation I landed on isn't ideal for this, but constructing edges based on finishing group would probably provide a more accurate picture when evaluating one cyclist against another.