Learning to identify quality articles on Hacker News

Ben Rudolph
Computer Science
Stanford University
brudolph@stanford.edu

Abstract

The goal of this project was to create a website classifier for Hacker News. Hacker News (news.ycombinator.com) aggregates articles and has users vote them up. If the article gets enough up votes in a certain amount of time, then it makes the front page of Hacker News which displays 30 articles at a time. Hacker News uses a ranking algorithm which leverages time since submission and number of up votes, which takes the form of:

$$Ranking = \frac{P-1}{(T+2)^G}$$

Where G is the gravity, P is the number of up votes and T is the time since submission.

If the ranking is among the top 30, then it is displayed on the front page. If an article gets displayed on the front page, then the site receives thousands of hits, so one can see it being beneficial to make it on the front page. This was the basic motivation behind my project. Given an article, could I predict if it was going to make the front page?

1. Introduction

Website classification has been done many times over on the Internet, but usually with more predictable categories. For Hacker News, there are a lot of intangible factors that go into making the top 30. For instance, a group of friends can force their article to the top 30 by all up voting the article themselves, and once the article has reached the top 30 it gets viewed many more times and thus has a greater chance of getting even more up votes. An article can usually make the top 30 if it gets 4 to 5 votes within an hour, so it's not implausible for a group to artificially push their article to the front page. Another factor to consider is luck. Not every article that is submitted gets read and an articleõs lifetime on the ñnew submissionò page is very short (depends on how many articles are being submitted during that time, but itõs usually around an hour). Theoretically, an article can be worthy of the top 30 but is only read by a small subset of the users that are interested in the topic. These are flaws in the Hacker News system itself, and thus will be very hard to predict when situations like stated above will arise.

2. Challenges

2.1 Coolness Factor

A lot of articles make the top 30 not because they have really good content but rather the author has made/coded a neat project and other people like it. As far as machine learning goes, it will be hard to define the usefulness or smartness of an application. One partial solution I came up with was identifying projects from non-projects. Some articles begin the title with 'Show HN:', and I used that as an indicator to tell if the article was a project or not.

2.2 Parsing Images/figures:

This was another stumbling block since if a document contained just a single image or figure, I had no way to get anything from the document. At one point I tried counting the number of images in the article but this was useless since it was hard to distinguish between pictures for ads and pictures that actually displayed information.

2.3 Identifying Trends

Dealing with trends in the technology world was another difficult challenge. Hacker News operates a lot on those trends. For example, when Steve Jobs died, all 30 articles of the front page were about him. Or sometimes something new in the tech world happened and that article made the front page. How to differentiate between an article about something that just recently happened and one that didn't was a tough problem to solve.

3. Approach

3.1 Logistic Regression

To tackle this assignment I decided to use logistic regression because, ultimately, I was more familiar with it and wanted to be able understand my results better. Once I got comfortable with my data using logistic regression, I tested my model on a support vector machine. One downside I found by using logistic regression was the time it took to run in comparison to the SVM. Since my project dealt a lot with text, I found myself having very long feature vectors which did not scale well with logistic regression.

3.2 Choosing a feature vector

As with most machine learning projects, choosing a good feature vector is paramount, so I decided to see if I could predict whether an article could make the front page. I setup a python script that tested me on the data that I had collected ï it would give me a link url and then I would input yes or no depending on whether I thought the article would make the front page.

I proceeded to open approximately 100 links and read the article. I took notes on the articles to try and prepare a usable feature vector. The first thing I noticed though was that it was actually very hard for me to predict accurately. Throughout the testing I found that keywords were essential to the likelihood of making the front page. Another feature I noticed was that almost all the articles came from the same pool of websites. While I was doing the testing, I became increasingly aware of the challenges I presented in the previous section. For example, two articles came from Github.com and they were both programming projects. One made it to the front page and the other one did not. As a human, I was able to tell which one was going to make it since I saw the practicality and usefulness in the program; however I wasn't sure how to represent this as a feature. Lastly, I noticed that some articles contained a lot of pictures while others did not, while this did not directly help me predict if the article was front page material or not, perhaps there was some sort of pattern that a machine could pick up on that I could not.

3.3 The Feature Vector

After much iteration, I ended up with this feature vector:

AskHN: <0 or 1> ï This indicated whether or not the article was a question to the Hacker News community or an article. Hacker News question were denoted by 'Ask HN:' in the title

ShowHN: <0 or 1> ï This indicated whether or not the article was a project built by the submitter. Often users would submit their projects with 'Show HN:' in the title to indicate that they built this website/product/application.

Hour: <0-23> ï This indicated the hour in which the article was submitted. This turned out to be a pretty important feature because during some hours there were a lot of submissions and it made the article less likely to make the front page.

WordCount: <Positive Integer> - This was simply the number of words in the article.

TitleWordVector: <Array of 0s and 1s> - This was a vector of length 400. I computed the 400 most frequent words that were in the title and if the article in question contained one of those words, then I would insert a 1, if not then a 0.

DocumentWordVector: < Array of 0s and 1s > - This was a vector of length 500, like the title vector, if the document in question contained one of the top 500 words then I would record a 1 in that position.

NounPhraseVector: < Array of 0s and 1s > - This was also a vector and was of length 500. It contained noun phrases from the document (had to have a length of at least 2 words). Same strategy used in previous vectors.

NamesVector: < Array of 0s and 1s > - This vector was 100 long and contained names from the article. Same strategy used as in previous vectors

ComheadVector: < Array of 0s and 1s > - Top 100 comheads/domains, if the article was one of the domains, then Iõd mark a corresponding 1 in the entry.

4 Data Preparation

4.1 Data Collection

To keep track of all the submitted articles I polled the stream of newly submitted articles every 5 minutes. I stored information on submission time, title, domain, url, and all the text in the article. I would also poll the front page of Hacker News every 5 minutes; if one of the titles that I had in my database was also on the front page than I would mark that example as having made the front page. This made for a very nice dataset, examples were all marked as either front page or not.

4.2 Data Pre-processing

Analyzing the text was another issue. The first obstacle to overcome was getting rid of all superficial text. It was easy enough to get rid of html markup using an html parser, but I only wanted to get text that involved the article. As a heuristic, I only took text from html blocks that had a 2:1 text to markup ratio. In other words, a markup tag like <p>Lots of text</p> would be selected but something like <a href=òwww.somepage.comò>ad</a> would not since there were more characters in the mark up than the actual text. This tactic helped filter out ad links and menus. However, I think it could have been improved even more if I created a machine learning algorithm for extracting important text from an article, but that's another project.

Never having taken a natural language processing class, this was the most challenging for me. Thankfully, Python's nltk (natural language toolkit) was very useful. After stemming and stopping all words in the text, I then broke the article into sentences. I then tagged all the words in the sentence with their part of speech. This served two purposes. One was to only count words that were nouns toward the word frequency vector and the other was to find noun phrases. To locate noun phrases I created a small regex which checked for a demonstrative followed by an optional adjective and then one or more nouns.

5 Results

5.1 Findings

After a slow start I was able to reach some conclusive results. See table 1 (last page) for a record of my results and their statistics (I didnŏt post accuracy since the data was skewed):

In the end, my F1 score hovered around 40-60 which I was happy with considering that there were a lot of cases I didnŏt know how to account for.

6 Future Improvements

6.1 Image Processing

As a possible extension, I wanted to somehow incorporate images into the feature vector. I attempted simply counting the number of images on a webpage, but this wasnŏt very helpful because of all the ads and other images that were not related. Perhaps developing a machine learning algorithm to decide whether an image is relevant to a document would help get a more accurate perception of how images contribute to the article.

6.2 Aesthetics

I also wanted to look into seeing how the aesthetics of a document affected whether people liked it or not. I was thinking about trying to determine the most used colors in a document to see if color affects readers in some unconscious fashion. Also the font might make a difference as well.

6.3 Text Analysis

As far as text analysis goes, I was thinking that as well as determining names in a document I could also determine products and companies using a list of current technology companies and products. Lastly I wanted to be able to distinguish between scholarly, fun, and blog articles. I think this too would require a machine learning algorithm, but it would be interesting to see if it helped classify front page articles.

6.4 Trends

Dealing with trends was a difficult issue and Iŏm not sure that only using the past 3 days of data was an adequate solution. For the future it would be interesting to add a sort of gravity to each feature. That is, given the time since submission somehow reduce the importance of its features. Iŏm not sure how I would implement this, but it might help better predict trends.

7 Applications

7.1 Web Application

I built a small web application (hnpredict.com) to apply the machine learning algorithm. It was interesting to build because it presented its own set of problems. Manipulating matrices in Octave is one thing, but storing them in a database and using them in a less mathematical language is a bit of a challenge. It proved quite useful to use a NoSql database like MongoDB to store my theta vector. Also, creating the feature vector given a website and URL was very time consuming. These are some things that I would need to take in consideration before implementing a machine learning algorithm for a web application. Itŏs impractical for real-world purposes to take a minute processing and creating a feature vector.

7.2 Other Applications

The machine learning algorithm to predict front page articles can be applied to other news sites other than Hacker News. Many websites like Reedit and Digg have much the same setup and could be analyzed in the same manner. I also could see this being used at news websites; given a bunch opinion pieces written by the outside world, they could then use the algorithm to help select which ones to be featured on the front page.

8 Conclusion

In the end, while I think my algorithm performed well considering the challenges it had to overcome, I donŏt think itŏs very practical. Itŏs too hard to predict whether an article is interesting and will often get confused on articles that are technology related but not very new or interesting. The predictions that it gets correct are often ones that are easy for the human to predict as well. Perhaps implementing some of the improvements mentioned in section 6 would help with this. As it stands I would consider it a negative result and would not use it on a regular basis to help me choose quality articles to Hacker News.

Trial Number	F1 Score	Precision	Recall	Description of Trial
1	0.0	0.0	0.0	Initial quick test. I didnt include noun phrases or names in this test. Tested on rather limited data too.
2	2.9	N/A	N/A	Created learning and validation curves. Found that the optimal lambda was around 50 and that I was drastically overfitting the data with all my features and not very many examples.
3	0.0	0.0	0.0	Added lots more data to both training and cross validation sets.
4	3.2	75	1.6	For this trial I switched to only creating my feature vector based on articles that were on the front page in hopes of defining what a good article is more concretely.
5	7.6	30.7	4.32	Didnt change anything except I started to experiment with changing the threshold. I moved the threshold down from .5 to .3.
6	14.8	8.7	50	Added noun phrases and name vectors! Had to drop my old database so I didnt have very many examples in this trial.
7	68.1	76.2	61.54	More data, and cut down the length of the document word and title word vectors.
8	64.7	61.11	68.75	Made a function to optimize threshold. I believe the reason the F1 score is falling is that Im beginning to have too much data.
9	25.0	20.83	31.25	Same data and features but tried running it on a SVM.
10	21	50.0	13.33	Even more data, I need to deal with the fact that articles that would make the front page 2 weeks ago would not make it now.
11	64.7	56.84	75.0	Only take data from the past 3 days to deal with trendiness of topics.